Мышление в стиле Visual Basic

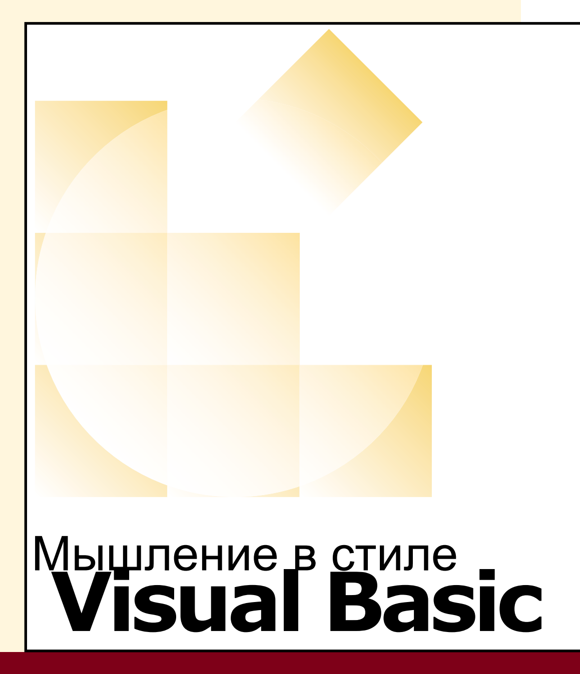
Настоящее издание рассматривает тему быстрой разработки Win-приложений средствами Visual Basic 6.0

В цикле упоминается домашняя страница автора:

http://www.vb.kiev.ua

Что здесь не рассматривается

- Базы данных,
- Компоненты ADO, DAO и т.д.



Введение

На вопрос, с чем ассоциируется у людей слово "Персональный компьютер" наверняка вы получите ответ: "Ну, Word, Windows". Это и не удивительно, - большинство программистов во всем мире первым ориентируются на именно эту операционную систему, так как именно она заполонила офисы и жилища. И наоборот, редкость программное обеспечение под Мак, Линукс, Юникс, "Ось Пополам", не говоря уже о полном отсутствии ПО для экзотических. И хотя сегодня ситуация немного изменилась в связи с растущим интересом к ОС Linux и другим "специфическим" операционным системам, доминирование "Окон" в среде обитания Человека еще долго будет бесспорным. Почему - тема отдельной готовящейся статьи.

* * *



Книга, которую Вы, уважаемый Читатель, держите в руках, имеет целью научить Вас писать программы для Windows95/98 на Visual Basic версии 6. Причем независимо от того, "чайник" Вы, или еще только собираетесь им стать, или же посредственный пользователь ПК. Может, Вы "эксперт" в области программирования для упомянутой Системы? Тогда читайте внимательнее - здесь будет рассказано о всевозможных трюках при помощи VB - языка, до последних пор не считавшегося языком программирования вообще.

Однако времена меняются, технологии не стоят на месте. Basic не должен был умереть. Четвертая версия громко заявила о себе как о версии, поддерживающей ООП (Объектно-ориентированное Программирование) и поэтому на порядок более значимой, однако в ней больше было ошибок, чем достоинств. Версия 5 доказала право на существования Бейсика. Еще оставались недоработки, но пятая версия уже расценивалась как профессиональный инструмент в области баз данных. Кроме того, была внедрена технология ActiveX, Бейсик сам "научился" создавать классные компоненты, Динамические Библиотеки и т.п. Версия 6 покорила многих и, наверное, развеяла сплетни о том, что это - интерпретатор. Нет. Это - полноценный компилятор. Причем не худший. Мало того, параллельно с Visual Studio, Microsoft внедрила VBA в Microsoft Office 8 (1997), CorelDraw! последовали их примеру, немного пополнив свой гибрид объектами Corel`a - работает медленно, но все ж быстрее, чем если бы Вы - "ручками". На Интернет-страницах стали появляться VBS - скрипты на Бейсике. Лишь Netscape упорно игнорирует этот факт - броузер до сих пор не понимает VBS. Пока…

Многие не находили лучшего аргумента, как большой размер исполняемых файлов и сопутствующего msvbvm60.dll. Однако сегодня ни один из языков программирования не позволит создать полностью независимое приложение под Windows с соответствующим интерфейсом. Можно "изголиться", конечно… А зачем?

Еще одни сетовали на медлительность. Однако исследования показали, что даже в случаях со сложными циклическими процессами разница в скорости между программами, написанными на Visual C++ и Visual Basic настолько невелика, что ее можно считать равной нулю. Я могу найти тысячу причин, по которым программа VC++ даст никудышние результаты в зависимости от различных условий. Скажите, у Вас всегда идеальные условия для запуска программ?. Мало того, приложения, использующие MFC, иногда просто "тормозят по полной программе".

В последнюю версию Visual Basic включены новые возможности в области Интернет (DHTML), Скриптинга, Баз Данных, и т.д. Почитайте What`s new - это интересно.

В конце концов, Бейсик всегда оставался Бейсиком. И хотя внутренности его "повзрослели" до неузнаваемости, писать на нем легко и приятно как в былые времена.

Прелюдия

Я бы мог начать статью словами: "Итак, начнем установку Visual Basic. Вставьте CD…" Однако это не входит в курс обучения "мышления в стиле VB". Главное в процессе инсталляции - внимательное чтение того, что Вам пишут. (Если пишут, значит это должен кто-то прочесть).

А начну я вот с чего. Начну с Правил Программирования Под Windows:

Перед тем, как начать проект (думаю, не нужно объяснять, что группу файлов, образующих связанный между собой Исходный Код еще нельзя назвать программой - а "проект" - то, что нужно) необходимо четко себе представить, КАК должна выглядеть будущая программа, что "уметь" и что должно получиться в результате ее работы - либо файлы - измененные, созданные, удаленные, либо Ваша программа - скринсейвер (та, что запускается по истечении, например, пятнадцати минут от начала ПОЛНОГО БЕЗДЕЙСТВИЯ пользователя. Бывает, возвращаешься с перекура, а на экране ничего не видно - только чей-то логотип летает).

По возможности внимательнее писать код, так как большинство ошибок в программах, даже коммерческих, уверяю Вас, - только по невнимательности программиста. Не раз бывало: вроде, все окей, - синтаксис в порядке - с ума сходишь в поисках ошибки. К тому же VB при тестовом запуске проверит синтаксис ключевых слов (лексем), их порядок расстановки, если он для них критичен и так далее. Тогда выводишь все это хозяйство на HP Laser Jet, берешь сигарету, и через минуту обнаруживаешь логическую ошибку.

Ну вот… проболтался. Да, ошибки бывают трех типов: синтаксические и логические. Как уже было сказано, синтаксические ошибки решаются просто: F5 (Тестовый запуск исходного кода), и логические. Такие ошибки могут годами присутствовать и портить Вам настроение при работе с несовершенным продуктом. Печально, когда логические ошибки встречаются в оборудовании (например, в Вашем процессоре Pentium).

Написание программы - дело не столько тонкое, сколько , я бы сказал, интимное… При детальном рассмотрении кода разных программистов можно даже обнаружить "почерк", стиль. Конечно, и типичные ошибки.

Так вот: возьмите себе за привычку начинать программы с Option Explicit - прямо в начале каждого модуля. Это избавит Вас от лишней головной боли. Ах, Вы еще не в курсе! Модулем называется файл (frm, bas, cls и др.), относящийся к Вашему проекту и, естественно, связанный с ним. По расширению файла сам Visual Basic знает, как его использовать, где и когда. Однако не думайте, что Вы его обманете, переименовав файл в Проводнике Windows. То, что Вы видите в самой среде разработки (Integrated Development Environment, IDE - в нашем случае это VB) - лишь надводная часть текста. Открыв Блокнотом файл, Вы обнаружите в нем еще много чего интересного, но пока непонятного, и от этого еще более интересного. Однако не делайте этого - Вы еще только начинающий программист. Ах, программистка? Тогда закройте IDE Бейсика и займитесь более интересным для Вас занятием.

Модули и компоненты

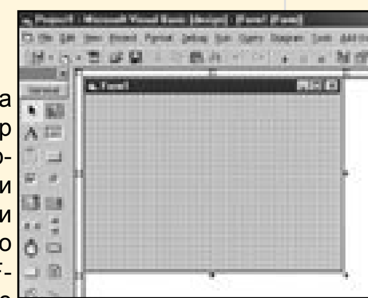
Итак, модули…. Нетрудно догадаться, что в состав проекта должен входить ну хотя бы один маленький модулечек. Какой - судить Вам. Для того, чтобы VB не наломал дров и не насовал Вам "под шумок" никому не нужной чепухи, при запуске он показывает вот такое окошко:

В одном случае будет создан один набор заготовленных модулей, в другом - другой.

Не мудрствуя лукаво, скажу: не морочьте себе голову - наша первая суперпрограмма будет стандартным исполняемым exe-файлом. Поэтому оставим то, что нам предлагают по умолчанию. То есть либо нажмем Enter, либо мышкой по OK. Теперь мы имеем в составе проекта лишь модуль Формы:

На рисунке видно, как кто-то пытается растянуть окошко по ширине и высоте. В этом режиме, который, к стати, в оригинале звучит как "Design Time", т.е. режим разработки, или Дизайна, пользовательского интерфейса - как Вам угоднее, - задаются основные визуальные и невизуальные характеристики для окошек, кнопочек, - всего того, без чего программа не программа. В число свойств формы, которые мы можем изменить на стадии разработки UI (User Interface) входит Caption.

Элементы управления



Палитра компонентов изображена на рисунке. Вначале доступен лишь набор элементов управления, "проживающих" в msvbvm60.dll. Чем хорошо ограничение только этими компонентами, так это малыми размерами инсталляционного пакета. Но разве можно устоять перед полнофункциональным RTF-редактором, позволяющим форматирование текста!? А MS InternetTransfer Control!? Ха! Вы еще не все знаете!!! Щелкните правой клавишей на палитре компонентов, из выпадающего меню выберите Components. Это - список компонентов, в данный момент предоставленных Вам Системой… Впечатляет? Многие из этих компонентов являются уже готовыми программами (в принципе, так задумано по определению компонентов ActiveX) и могут наделить Ваше произведение потрясающими возможностями, - прямо как именитые релизы коммерческих компаний. А Вы знаете, что каждый раз при инсталляции нового чужого программного продукта этот список пополняется и, возможно, "тот" программист догадывался, что Вы ужасно любите копаться в списке компонентов и сделал Вам подарок - не защитил свой ActiveX Control лицензией. Тогда Вам остается только выбрать его из списка и...

Можно использовать на страничке в Интернет. (Однако не расчитывайте на безнаказанное вредительство в отношении доверчивых серферов - броузер пользователя предупредит несчастного о возможных тер-рактах перед Install-On-Demand - Инсталляцией-На-Лету. Хотя…) Можно в программе. Можно в документе MS Word 97, Excel, Access. Можно программу написать в виде такого компонента и в следующий раз для решения похожих задач сэкономить пару недель времени - просто выбрав свое творение в палитре. К счастью, начиная с версии 5.0 такое возможно. А можно продавать и со временем сделать на этом миллиарды! Не верите? Спросите у разработчика MS Visual Basic. Да! И самое главное - готовые компоненты (файлы с расширением OCX) можно использовать в любых языках программирования, поддерживающих технологию ActiveX. К ним относятся Visual C++, Java, Delphi…

Для нормальной работы любой элемент управления должен быть зарегистрирован в Системе - мало одного лишь факта нахождения его в системной директории. Обычно это удел Инсталляторов (Wise, InstallShield, SaxSetup etc.). В состав Visual Studio входит Package and Deployment Wizard (Мастер упаковки и распространения). Вещь нужная, однако гибкость его оставляет желать лучшего. Шаг влево-шаг вправо - инсталляция идет под откос. Поэтому рекомендую инспользовать InstallShield Express for Delphi 5+. Интересно, что к Delphi можно отнести лишь первую вкладку, выясняющую, что Вы использовали в проекте, остальное - для всех, в том числе и для VB. Таким образом можно приловчиться составлять инсталляции только для OCX-файлов.

Далее будет рассказано, как зарегистрировать компонент при помощи Вашей программы.

Свойства элементов управления

Любой компонент ActiveX имеет свойства, методы, события. Некоторые из них доступны в режиме разработки (Design Time). Некоторые - только в ходе выполнения, - например, SelLength текстового поля означает длина выделенного текста. Другие свойства являются только для чтения. Например, изменить свойство OSVersion элемента управления SysInfo, скорее всего, удастся далеко не каждому…

Возьмем типичный элемент управления (Control) - TextBox. В принципе, понятно, как он работает, где его применяют, и чего от него ожидать.

Свойства - интереснейшая штука. Можно написать целую программу благодаря фокусам с одними только свойствами. Свойство и имя самого объекта - носителя этого свойства разделяются точкой (и ничем другим), После того как пользователь наклацал текст в текстбокс (поле ввода), его свойство Text стало равным тексту в этом поле. Логично…

Например:

```
TextBox1.Text = "Текст, который написал Я!"
```

Имена элементов управления - вещь нехитрая: главное, чтобы они:

- не содержали нелатинских символов (жїїцщ и т.д.),
- не содержали спецсимволы (!"№;%:?*(-)=\, .>;
- не начинались с цифры (1LockDrive)
- были длиной не более 256 символов.

При добавлении, например, текстового поля в нашем случае ему будет назначено имя TextBox1. Зачем нужна нумерация? А затем, чтобы избежать повторения имен - одинаковых-то не должно быть! Следующий текстовый бокс будет уже с цифрой 2, следующий - 3 и так далее. Поэкспериментируйте. Нажмите клавишу Control, и, не отпуская, пощелкайте двойными щелчками на эмблемке текстового поля - это удобнее, чем каждый раз щелкать на панели инструментов, а затем на форме. Закончив, порастаскивайте их в разные стороны - они ведь наложились одно на другое.

Помните: чем больше у вас на форме элементов, тем больше памяти "съест" Ваше произведение… Однако есть способы избежать этой проблемы - с помощью массива. Но об этом - несколько позже.

У каждого элемента управления есть свойство по умолчанию. Это несколько упрощает нам жизнь.

Для показанного на рисунке элемента управления таким свойством является Shape (геометрическая форма), так как на самом деле он не предназначен для управления чем-либо - как и у элемента управления Line (линия), его функции - чисто оформительские.

Как узнать, какое свойство является "умолчательным" для выбранного элемента?

Есть хитрый способ.

Для этого нам необходимо поместить на форму кнопку. Не важно, что на ней написано, - это сейчас не принципиально, но если Вас это смущает - на панели свойств найдите Caption и замените его на что-нибудь более информативное.

Дважды щелкните на кнопке, которая уже помещена на форму. Откроется Событие по умолчанию. Для кнопки это Click (щелчок). А что, можно было придумать что-то другое?

Бейсик кое-что сделает за Вас:

```
Private Sub Command1_Click()
    '
End Sub
```

Очевидно, что он разделил имя элемента управления и событие символом подчеркивания. Дело в том, что, будем говорить так, в Кнопке1 "зашита" процедура, отвечающая за событие Клик. И пишется она именно с символом подчеркивания. Естественно, если этот символ удалить, кнопка не догадается, что к ней обращаются.

Как Вы могли заметить, курсор остался между этих двух строк. Это значит, что VB ждет, что вы продолжите именно с этого места.

Добавим всего лишь одно слово:

```
MsgBox
```

Теперь добавим пробел.

О регистре букв можно не беспокоиться - Visual Basic побеспокоится об этом сам. Как только курсор перейдет на другую строку, все станет на свои места. Но это не единственное его достоинство перед другими языками программирования. Он еще и заканчивает Ваши программные фразы! Кроме того, он постоянно подсказывает Вам, что после чего писать, когда дело касается встроенных в него функций, приемов. Окно сообщения (MsgBox, от Message box) - яркий тому пример. Вот какую строку он пишет под курсором:

```
MsgBox(Prompt, [Buttons As VbMsgBoxStyle = vbOkOnly], [Title], [HelpFile], [Context]) As VbMsgBoxResult
```

По мере написания кода для этой функции активизированные поля выделяются жирным. На этом примере видно, что Бейсик готов принять значение Prompt, что дословно можно перевести как "Приглашение". Далее - Buttons (кнопки) плюс квадратные скобки, которые означают что параметр необязателен. То есть, его можно не писать, - просто поставьте запятую, показав тем самым Бейсику, что прошли этот параметр. Title означает заголовок окна сообщения. Если его игнорировать, пользователь Вашего приложения наверняка увидит нечто типа "Project1". Исправить положение можно двумя способами:

Зайти в меню **Project->Project1 Properties**…, отыскать

Project name… или приказать приложению менять это свойство программы (*App.Title*) на другое. В этом случае используем событие для окна:

ДваждыКлик по форме. Открывается соответствующее событие (оно же - по умолчанию), в нашем случае - Form_Load, курсор - в нужном месте - все готово к вводу текста:

```
App.Title = Now
```

Обратите внимание - *Now* без кавычек!

Хорошо, мы оставили кнопки такими, какие установлены по умолчанию. А как узнать, какие именно?

Если присмотримся, увидим следующее:

```
 = vbOkOnly
```

Отсюда следует что будет показано окно ТолькоСКнопкойОК

Кнопки бывают такими:

VbOkOnly

VbYesNo…

Стоп! Сделаем проще: после строки сообщения (Prompt) в прямых кавычках, (строчки текста без прямых кавычек Бейсик рассматривать упорно не желает) поставьте запятую - IDE Visual Basic Вам покажет целый список значений для параметра Buttons (Кнопки). Такие параметры, которые он показывает в списке во время введения кода с "клавы" называются Перечислимыми. На самом деле за более-менее удобочитаемыми терминами типа vbOkOnly, vbAbortRetryIgnore (пример: попытайтесь прочитать что-нибудь с флоппи-дисковода, не вставив диск) кроются числа - от нуля до максимального. Нулю соответствует первый элемент в списке перечислимых данных.

Просто щелкните дважды мышью на элементе списка, который Вам больше по душе.

В итоге получим строку:

```
MsgBox Shape1, vbOkOnly
```

Остальные параметры для MsgBox тоже в квадратных скобках - значит, о них можно не беспокоиться. Оставим это более опытным программистам

Теперь найдем на панели вверху кнопку play с такой вот пиктограммкой "Play" и кликнем на ней. (Того же можно добиться, нажав F5).

Программа запустилась. Нажмем на кнопочке и получим сообщение : 0

Значит, свойство по умолчанию обладает целым списком возможных значений. Вооружившись логикой и немного - интуицией, можно предположить, что таким свойством должно быть Shape для одноименного элемента. Так, для поля ввода это Text, для ярлыка - Caption (надпись), для таймера - Enabled (активен или нет), для Списка директорий и списка файлов - Path и Filename соответственно. Исключение составляет элемент управления Form (наше с Вами окно), для которого нет свойства по умолчанию - есть только событие - Load, которое происходит во время загрузки формы.

Таким образом, исполнив код MsgBox Text1, мы получим сообщение в виде текста из текстового поля ввода (свойство Text).

Кажется, немного разобрались с MsgБоксами. Большинство книг предлагает читателю *ПервуюПолноценнуюПрограммуДляВиндоуз* "

```
Hello, World".
```

Мы же в качестве "пробы пера" узнаем, куда проинсталлирована наша ОС. И знаете что? Я наверняка кого-нибудь удивлю сказанным, ибо процедура сия требует усилий немалых и опыта большого… Потому как на всех Интернет-конференциях тема эта относится к "трюкам" и "ловкачеству" с вызовом API-функций. (Для нетерпеливых: API-функции (Application Programming Interface) "вшиты" в динамические библиотеки Windows - DLL-файлы. API-функциями можно пользоваться во всех современных языках программирования. Носители таких функций - системные файлы Вашей Windows95/98. Например, стандартное окно Save As… - функция, находящаяся в Commdlg.dll и Commdlg32.dll). Длина кода с такими "извращениями" может достигать пару машинописных страниц и под силу только волшебникам. Однако мы обойдемся лишь одной строкой кода:

```
MsgBox Environ$(4)
```

Переменные

Конечно, чтобы использовать эту строку в программных целях, ее нужно немного обрезать слева. Для этого нам не обойтись без "временного хранилища" для данных - переменной. Такое хранилище после операции со строкой Вам больше не понадобится, поэтому можно сделать его локальным:

```
Dim Var1 As String
```

As String означает, что информация, хранящаяся в этой переменной, представлена Как Строка.

Если Вы считаете, что строку типа winbootdir=C:\WINDOWS можно повторно использовать в этой же программе - вместо Dim напишем Public. Понятно, что при этом переменная стала Публичной, то есть доступной всем отовсюду.

Переменные могут быть доступны из других модулей. Например, в Form1 объявлена переменная Var1, содержащая строку winbootdir=C:\WINDOWS. А проект содержит два окна. Чтобы узнать, какую строку содержит Var1, для процедуры события "клик" нужно ввести код:

```
MsgBox Form1.Var1
```

Если Вы посчитали ее нужной и написали Public Var1 As String, все будет окей и ваша программа будет классно продаваться, но если Dim, да еще и объявили ее прямо в процедуре события какого-нибудь Click`a - Бейсик не даст Вам даже откомпилировать код с криком: "Переменная отсутствует", или что-то в этом роде. И будет прав - все дело в области видимости.

Самые "дальнобойные" переменные лучше объявлять в Стандартных модулях. (Конкретно о модулях читайте в следующих уроках). Потому как у такого рода модулей всегда приритет по сравнению с модулями форм.

Итак,

```
Dim Var1 As String
Var1 = Environ$(4)
Var1 = Right(Var1, Len(Var1) - 10 - 1)
MsgBox Var1
```

Понять урезание строки справа поможет своего рода транскрипция:

```
[Переменная] = Справа([ТаЖеПеременная], _
[ЕеДлина]) - [ДлинаНенужногоФрагмента] - [ОдинСимвол]
```

Зачем отнимать еще один символ? Затем, чтобы исключить попадание последнего символа ненужного нам фрагмента текста, а именно "=". Теория без практики мертва - экспериментируйте.

Интересно? - попробуйте также Left.

Я рад, если такое "разжевывание" кому-то помогает.

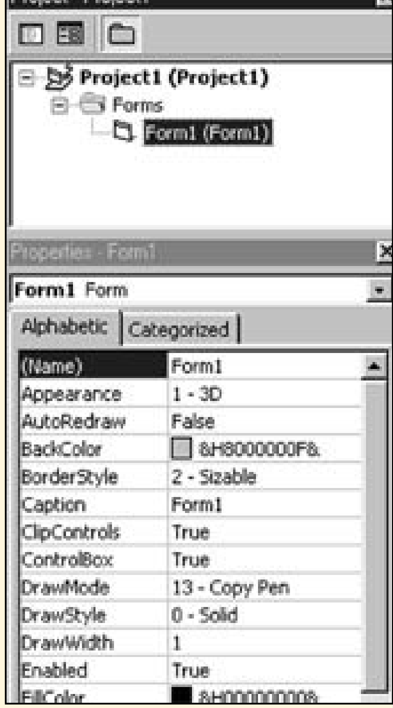
Интересен тот факт, что Microsoft Word выполнит эту операцию с таким же успехом! (Подсказка: Alt+F11, ввести MyProcedure, Enter (и тут Джин завершил за Вас блок процедуры), MsgBox Environ$(4), F5). Все просто - в Microsoft Word встроен язык программирования Visual Basic for Applications (VBA), который поддерживает многое из того, что "умеет" обычный VB из пакета Visual Studio 6.

В данном случае встроенная в Visual Basic функция Environ$ получила от нас аргумент - цифру 4. А что, если указать 3? А 5?

Ну, каждый раз запускать программу, писать новое значение, останавливать... Не лучше ли вводить нужное значение в текстовое поле, чтобы программа могла узнавать его в ходе выполнения после нажатия пользователем Enter?

Окей, и это не сложно:

Если Project Explorer не активен, его можно вызвать клавишами Ctlrl+R или меню View->Project Explorer. В окне Project Explorer`a всегда можно доступиться к любому компоненту проекта - окну, модулю, другим файлам, например, Readme.txt. Перейдем в редактор Объекта (View Object). Для этого нажмем кнопочку с изображением символического окна (См. рис. справа).

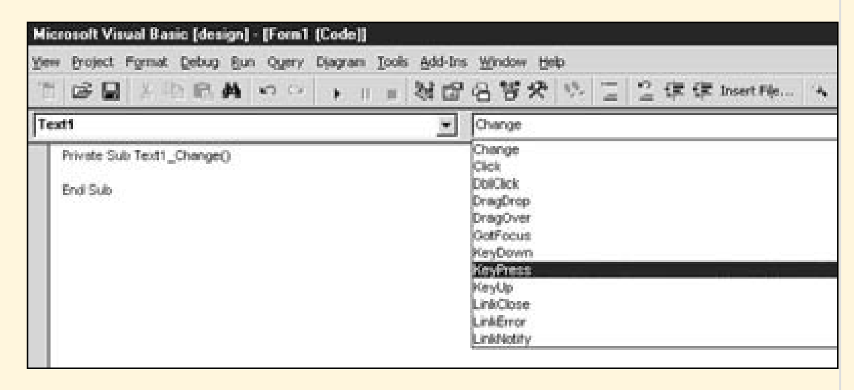


Еще один способ - дважды щелкнуть в этом окне по Form1.

Удалим кнопку Command1 с нашей формы - она нам больше не нужна.

Добавим текстовое поле на форму. Оставим его имя таким, как его назвал Бейсик. Дважды щелчок - и Вы в редакторе кода.

По умолчанию нам предлагается процедура Text1_Change (Изменение); заменим ее на более интересную - KeyPress (НажатиеКлавиши) как показано на рисунке.



Следует заметить, что такой способ является альтернативным для создания процедур обработки событий для элементов, использованных в Вашей программе. Если выбрать из левого списка (там где указано

поле ввода TextBox1) Form, VB автоматом подготовит все необходимое для процедуры по умолчанию для формы.

Итак, введем текст между Text1_KeyPress(KeyAscii As Integer) и End Sub:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
  If KeyAscii = 13 And Text1 <> "" And IsNumeric(Text1) = True Then
  ' Если клавиша - Enter,
  ' и в текстовом поле
  ' имеется текст,
  ' и при этом это число, то:
  KeyAscii = 0 'Запретим добавлять
  'символ абзаца в текстовое поле
  MsgBox Environ$(Val(Text1)) 'Val переведет
  'содержимое в другой тип
  'данных - Integer
  If Environ$(Val(Text1)) = "" Then ' Если вернулось
  ' "пустое" значение, то:
  Beep 'Издать характерный бип
  Exit Sub ' Выйти из процедуры
  End If
  Text1 = "" ' Очистить поле ввода
  End If
End Sub
```

Как видно, программа проверяет данные. Операторы If, Then можно дословно переводить - вот ТО САМОЕ они и значат. Правда, мы не использовали дополнительный оператор Else (иначе), однако программе, описываемой в следующем номере, без него не обойтись.

С операторами проверки условий, казалось бы, проблем быть не должно, однако большинство начинающих (и не только!) программистов путается в нагромождении/ветвлении этих самых Если-То-Иначе. Поэтому - совет: сразу после If допишите всю структуру, например:

```
If Text1 <> "" Then

Else

End If,
```

затем вернитесь и повписывайте что куда нужно.

Эпилог

В следующих уроках мы детально рассмотрим разницу в процедурах функциях, событиях, рассмотрим типы данных, создадим более-менее "продвинутый" и очень удобный текстовый редактор. Не обойтись без выбора шрифтов и сохранения текста на диске. Кстати, готовое приложение можно назвать Notepad.exe и закинуть в директорию Windows… К тому же придумать свой формат файлов и связать его с новым приложением…Я также научу читателя, как сделать его редактором Интернет-страничек по умолчанию.

Процедуры и функции

В ходе написания кода "затяжного проекта" зачастую возникает потребность в использовании определенных готовых фрагментов, за исключением лишь некоторых деталей, например, при обработке другой переменной или любого другого программного обьекта. Вот пример чтения текстового файла построчно с выведением каждой строки в окне сообщения:

```
Dim i As Integer
Dim strVariable As String
   i = FreeFile
   Open "c:\Test\Tutor-001.htm" For Input As #i
   While Not EOF(i)
      Line Input #i, strVariable
      MsgBox strVariable, vbOkOnly+vbExclamation, _
           "Please read the line shown below"
   Wend
```

Может так случиться, что понадобится такое же построчное считывание текстовых данных, но без выведения пользователю информации или, например, по отношению к другому файлу. Тогда пришлось бы сдублировать код, пропустив

```
MsgBox strVariable, vbOkOnly+vbExclamation, _
"Please read the line shown below"
```

и заменив

```
Open "c:\Test\Tutor-001.htm"...
```

на

```
Open "c:\Test\Tutor-002.htm"
```

Можно, конечно, поступить и таким образом, пожертвовав при этом размерами Вашего EXE (который вмещается разве что на CD), удобством последующего редактирования программного текста, (а это неизбежно как и то, что за осенью следует зима) во имя простоты и примитивизма. Так, наверное, поступали бы все программисты, если бы в Visual Basic не было процедур и функций.

Процедуры и функции - весьма удобные "фишки" любого, даже дореволюционного, языка программирования. Оба они во многом схожи, однако кардинально отличны их цели и области применения. Следует запомнить: процедура - штука достаточно примитивная - выполнил и забыл. С нее, в общем-то, нечего и взять, и применяют ее в соответствующих. Например, процедура WhatTimeIsItNow в любом случае будет выполнена, если только не наступил Конец Света и Время не остановилось:

```
Private Sub WhatTimeIsItNow()
   MsgBox "Today Is " & Date
End Sub
```

На самом же деле предпочтение процедур функциям должны диктоваться спецификой Вашего кода, - сиюминутные советы здесь вовсе неуместны.

Обработка ошибок в процедурах

Конечно, ошибки возможны даже в самых неожиданных участках кода и мастерство программиста - предусмотреть наиболее вероятные из них еще на стадии разработки. Например, выйдет новая ОС, не поддерживающая Date, Time или Now.

К стати, с учетом всевозможных затруднительных ситуаций и логических тупиков опытные программеры используют нечто эдакое, при котором программуля игнорирует дальнейший код и аварийно прекращает процедуру. При этом она просто-напросто сообщит, что процедура не выполнена. Возможно, она предоставит пользователю характерные варианты решения проблемы.

Да, VB располагает несколькими операторами, благодаря которым программа "срезает угол" и ретируется, обходя "узкие места":

```
Private Sub ReadHTML()
   Dim i As Integer
   Dim strVariable As String
   i = FreeFile
   On Error GoTo FileNotFound
   Open "c:\Test\Tutor-001.htm" For Input As #i
   While Not EOF(i)
         Line Input #i, strVariable
         MsgBox strVariable, vbOkOnly+vbExclamation, _
         "Please read the line shown below"
   Wend
   Exit Sub
   ' Если ексцессов не произошло,
   ' на этом Sub "сворачивает удочки".

   FileNotFound:
   Beep
   MsgBox "Файл не найден, однако."
   Exit Sub
End Sub
```

Оператор On Error означает "в случае ошибки". GoTo - "Перейти К" [метке]. Однако бывают такие ошибки, когда просто перейти к метке оказывается либо малодейственным средством, либо нецелесообразным, либо вообще бессмысленным. Например, программа-фикция для восстановления поврежденного файла в случае отсутствия такового преспокойно может продолжить работу, переспросив пользователя об имени файла и пути к нему, а вот программа, натолкнувшаяся на более серъезную проблему (например, невозможно создать экземпляр элемента управления), аварийно заканчивает работу. В таких вот сложных условиях следует писать несколько иначе: On Error Exit Sub

Если же пользователь может продолжать нормальную работу, ничего не подозревая, лучше написать так: On Error Resume Next, после чего будет исполнена следующая строка кода.

Заметьте, перед меткой FileNotFound: мы установили оператор Exit Sub. Он, в частности, и делит наш код на две части: "обычную" и "экстремальную". Уберите его, и Вы каждый раз будете получать сообщение об ошибке даже после успешной обработки файла. И наоборот, очередь до строк под FileNotFound дойдет лишь в случае ошибки при Open "c:\Test\Tutor-001.htm" For Input As #i (например, на диске по указанному пути не окажется файла).

Как уже было сказано, процедуры, в отличие от функций, никакой информации нам передать не могут, - их применение - в самых тривиальных ситуациях, выполнение которых не требует заполучения никакого результата (в смысле значения какой-либо переменной).

Но в том случае, когда мы ждем от программы каких-то конкретных данных, целесообразно применять функции, причем чем большим количеством функций Вы автоматизируете свое творение, тем приятнее и, что важнее для программиста любой квалификации, проще писать. Неужели не выглядят заманчивыми функции типа:

```
varFloppySpace = GetFloppyFreeSpace("a:\"),
varLinks = GetAllHyperlinks("c:\Test\Tutor-003.htm", True, False)
```

или

```
varPassword = HackZipPassword("d:\ISP_Passwrds.zip")?
```

Действительно, достаточно функции GetAllHyperlinks передать имя иного файла, и она обработает именно тот, иной файл, если функции GetFloppyFreeSpace указать имя другого диска, например, "b:\", то юзер получит свободное место на диске b:\

Вызов функции с передачей ей аргументов выглядит следующим образом:

```
MyVariable = GetFloppyFreeSpace(a:\)
```

Ваша функция GetFloppyFreeSpace вернет Вам необходимую информацию, а именно количество свободных байтов на флопе. Но в каком виде?

Если присмотреться повнимательнее в первую же строку обьявления функции, то за скобками мы увидим тип данных, возвращаемых ею. Причем передавать ей можно только указанный в ее скобках тип данных. В нашем случае с гибкими дисками - String, что значит "строка". Иначе мы получим ошибку несоответствия типов. А возвращает функция некое число типа Long, (приблизительно равное четырем гигабайтам).

К стати, Бейсик готов к тому, что функции передан(ы) аргумент(ы), тип которого/ых не определен - Вам безразличен тип данных - у него на это свой ответ - тип Variant, эдакий хамелеон, приспосабливающийся к данным и принимающий их тип. Оно-то, конечно, удобно, если не считать, что это - самый "прожорливый" тип. Ах, Вы ожидаете увидеть свой софт быстрым и "легким"? Тогда, по возможности, не используйте тип Variant, не то будет Ваша программа такой же быстрой, как Win2000 на Pentium-100.

С процедурами все проще. Главное - не забыть, как она зовется. Черканул Call WhatTimeIsItNow - и дело сделано.

Следует заметить (и это очень важно), что в вызовах процедур типа Call WhatTimeIsItNow мы вполне обходимся без скобок (да и в обьявлениях процедур эти скобки пусты), но в вызовах функций мы обязаны в скобках указать аргументы.

Вот пример гибкой реализации "форточной" процедуры MsgBox:

```
Call MsgBox "Сообщение. Кликни, - я исчезну.", _
    vbOkOnly+vbExclamation, "Пример коробки сообщения"
```

(обратите внимание на **Call** - вызвать. Для функций этот оператор не применим! Впрочем, для внутренних процедур VBA Call можно пропускать).

А вот ее "двойник" - функция:

```
Dim ButtonsIPressed as Long
ButtonsIPressed = MsgBox("Ну-ка, выбери кнопочку!", _
           vbYesNoCancel+vbCritical, _
          "Пример коробки сообщения ")
  Select Case ButtonsIPressed
  Case vbYes
  MsgBox "Ты нажал Да"
  Case vbNo
  MsgBox "Ты нажал Нет"
  Case vbCancel
  MsgBox "Ты нажал Отмена"
  Case Else
  MsgBox "Сорри, по-моему, плохо работает кулер...", _
       vbCritical, "Ooooopsss…"
  End Select
```

Теперь, надеюсь, понятно, почему у функции InputBox (окно ввода) нет такого же "двойника"? Верно: единственный смысл в InputBox - выпытать у пользователя некую строчную информацию.

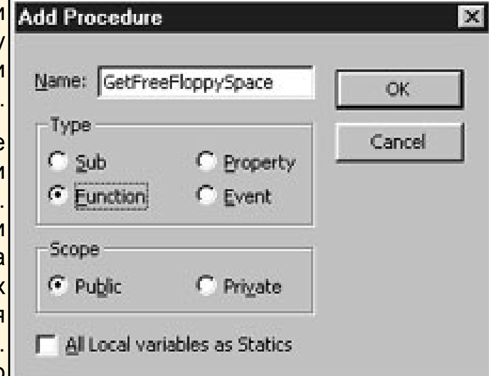
Обработка ошибки в функции

Кроме перечисленных "фишек", прелесть функций состоит еще и в обработке ошибок, - хотя и экзотической, но весьма эффективной: предположим, пройдя часть кода функции, программа "напоролась" на непредвиденную ошибку, и оставшаяся часть кода осталась невыполненной. По сему логично считать попытку провалом:

```
Private Function FloppyInserted(DiskName = "a:\") As Boolean
   ' Самый простой способ проверить,
   ' воткнута ли дискетка - попытаться писануть на нее.
   FloppyInserted = True 'Предположим, все ОК.
   Dim i As Integer
   i = FreeFile
   On Error Goto NoDisk
   Open DiskName & "TestFile.txt" For Output As #i
   Print #i, "Вполне исправная дискета!"
   Close #i
   Kill DiskName & "TestFile.txt"
   FloppyInserted = True
   ' Раз уж выполняется
   ' данная строка, значит
   ' диск - в дисководе.
   Exit Function
   NoDisk:
   ' Эти строки будут выполнены, если произойдет ошибка
   ' при выполнении Open DiskName & "TestFile.txt" For Output As #i
   FloppyInserted = False
   ' Данный код обработки ошибки обращения к диску
   ' не избавляет нас от необходимости выхода из функции.
   ' Однако мы успели присвоить функции значение "Ложь",
   ' чего и требовалось узнать.
   Exit Function
End Function
```

Код, приведенный выше, не совершенен - здесь любая ошибка (нехватка места, защита гибкого диска от записи и т.п.) будет расцениваться как отсутствие диска в дисководе, однако окажется весьма полезным, если использовать его в целом комплексе мероприятий типа ReadWriteAccess("a:\"), GetFreeDiskSpace("a:\") и тому подобных.

IDE Бейсика предусматривает более-менее удобный способ создания функции или процедуры. Через меню **Tools->Add Procedure** доступно диалоговое окно, в котором наглядно представлены параметры и свойства создаваемой Вами функции/процедуры:



Но это - не единственный способ создания функций или процедур. Некоторые программисты вопреки всему набирают их "руцями", точно так же, как веб-мастера - свои страницы (в традиционном html-редакторе "Нотопад.Экзе").

Открывая программный код "крутых" программистов и не очень, натыкаешься на несметное количество функций и процедур, которые вызываются из разных мест программы. Некоторые деятели vb-движения являются явными приверженцами ООП, другие - нет. Первых так и тянет на создание классов, коллекций и подобных вещей, их удобных событий и свойств. Вторые довольствуются "стандартными" модулями (с расширениями .bas). Действительно, при написании простого текстового редактора совсем не обязательно писать громоздкий код, изобретая класс clsDoc. Хотя и здесь могут

быть разночтения. Как говорится, лучше год потратить на класс, зато за пять минут написать программу. Однако классы и ООП - отдельная широчайшая тема, - интересная и захватывающая, ведь это - "конек", приобретенный Бейсиком начиная с версии 5. Классы можно использовать не только в других проектах, но и сдавать в аренду другим - чужим программам! А не доводилось ли Вам видеть на лоне Интернет-страницы нормальный документ MS Word, прямо-таки "вживленный" в нее? Так вот: из VB-скрипта создан экземпляр класса документа Word - всего-то! Однако слишком хорошо - тоже не хорошо.

Быль#1. О двух программерах

Жили-были два программиста. Оба опытны, умны и безукоризненно знали Visual Basic, так как он им был родным языком. Но один знал ООП и пользовался им как мог, а другой - тоже знал, но недолюбливал. И задала им злая фея задание - всем одно и то же. Вот и принялись они выполнять один и тот же проект различными средствами: один - основываясь на ООП, другой - на обычные "процедурности". Последний напихал в приложение всего, что только можно было, и был код его в 30 Мб в зипе, но написанный за два дня, а первый игрался с классами два месяца, но, опоздав, сотворил шедевр.

Что лучше? Не знаю.

Со "стандартными" модулями дела также не столь плохи: мало того, что они также могут многократно использоваться либо Вами, либо сторонними программистами, например, не желающими знать механизм обращения к Реестру, но отчаянно желающими натворить гадостей пользователям программы.

Часты случаи распространения модулей через Сеть Интернет, причем такие модули считаются "конечным" продуктом и, в принципе, не подлежат переработке, за исключением случаев выхода новых версий компонентов ОС, к которым эти модули обращаются. Яркий пример - размноженный миллионными тиражами "бестселлер" Winsock.bas. Я же планирую разместить некоторые из модулей, рассматриваемых в следующих статьях цикла, на сайте, потому как из-за их "неприличного" объема они не могут быть опубликованы в нашем издании. Модули и исходный код для статей можно будет свободным порядком "грузить" и использовать в любых целях - как коммерческих, так и образовательно-поучительных.

Не расслабляйтесь, это еще не все: устраиваться на работу программистом рановато.

Существует такая радость, как область видимости (scope). Локальная (Private) функция или процедура, написанная в модуле формы, к тому же не в разделе General Declarations, то есть наравне с процедурами кликов, драг`н`дропов и им подобных реакций компонентов на действия юзеров, становится абсолютно недоступной для других форм. Например, где-то в Form1 у Вас есть *Private Function LoadFonts (ListToFill = List1) As Boolean*, а на второй форме (Form2) есть кнопка "Заполнить шрифтами список". Она не сработает, потому что LoadFonts есть приватная, а не Public. А публичные функции пишутся ТОЛЬКО в разделе глобальных обьявлений модулей форм (это на самом верху). Тогда вполне законным является код: Call Form1.DrinkBeer.

В противном случае Бейсик просто "психанет" и не даст Вам запустить программу.

Разместив же код полезной функции/процедуры в "стандартном" модуле, Вы навеки избавлены от необходимости указывать форму/модуль-носитель. К сожалению, множество ошибок, выявленных в процессе дебаггинга, связаны с недоработками именно в этой области: программер рассчитывает на одно, а на практике - все точно

так с точностью до наоборот. Объясню: абсолютно все переменные, объявленные в проекте, имеют также свою область видимости. В случае, если "стандартном" в модуле (или в модуле класса, но об этом - позже) содержится переменная var1 и в форме - такая же, (при этом первую принято рассматривать как "козырную"/глобальную/публичную), то... приоритет у локальной. Хм...

Подсказка: Нужно использовать другую, прописанную в модуле? Пишите так: Module1.Var1

Быль#2. Немного об оптимизизации кода

Функции - удобная вещь. Недаром же это основная часть языков программирования.

Моя первая программа должна была стать очень удобной для редактирования и весьма "продвинутой" - впервые ощутив мощь лаконизации кода средствами многократно используемого кода (проф. термин: reusable code) в лице глобальных функций и процедур, я был поставлен практически в тупик, когда весьма ощутимыми оказались проблемы слежения за значениями некоторых важных переменных, программа "висла" непонятно в каком месте и непонятно от чего, в процедуре загрузки главной формы покоились десятки вызовов других процедур инициализации, загрузки Ресурсов, проверки настроек - одни - в Реестре, с этим работала одна функция, другие - в INI - это было по части другой (третья собирала все в кучу и передавала еще одной после проверки в четвертой), и так далее, которые, в свою очередь, тоже являлись всего лишь контейнером (для простоты) с другими вызовами функций, находящихся в "стандартном" модуле и потому "видимом" для других модулей, однако не все функции и процедуры я поместил в эти чудо-модули - некоторые из них были актуальны только для тех модулей, в которых они были объявлены. Был также создан класс, пользовавшийся и теми, и другими плюс своими, и это естественно.

В конце концов мне таки пришлось пересмотреть свои взгляды: зачем мне такая простая программа?

Ну, вот мы и разобрались с этими чудовищами - функциями и процедурами. Надеюсь, эти два термина не будут приводить Вас тихий ужас при необходимости их использования. Внимательный читатель готов к созданию текстового редактора на основе многократно используемого кода.

Текстовый редактор

С чего начинается создание программы, даже самой примитивной? (Подразумевается, что основные правила подготовки к созданию проекта, а именно определение цели программы, ее логического устройства и общей концепции с результатами программист уже для себя уяснил). Конечно, с интерфейса пользователя. Это - своего рода "рабочая поверхность" сложного механизма. Подавляющее большинство книг, выпущенных в свет за последние пяток лет, твердят одно и то же: перетащили кнопку в форму, перетащили туда же элемент управления (далее - ЭУ) "ярлык", или "лейбл", или "надпись" - кому как удобнее (в оригинале это Label), - и все! Простейшая 32-разрядная программа готова! Да, такую программу можно откомпилировать, (без отладки в данном случае можно обойтись), создать дистрибутив (еще раз рекомендую как минимум InstallShield Express for Delphi, где к Delphi относится только вторая вкладка. Самый "продвинутый" читатель может воспользоваться InstallShield Pro, обладающим собственным языком программирования и объектной моделью. Зайдите на сайт поддержки разработчика дабы убедиться в его мощи). Далее - реклама, ну и … пожинание плодов славы и отдачи от реализации бестселлера.

Итак, я не стану ломать вековые традиции, а посему мы начнем в том же духе: запустите IDE Бейсика, и в диалоге New Project выберите Standard EXE. В общем-то, нужно только нажать Enter - VB по умолчанию предлагает именно такой проект.

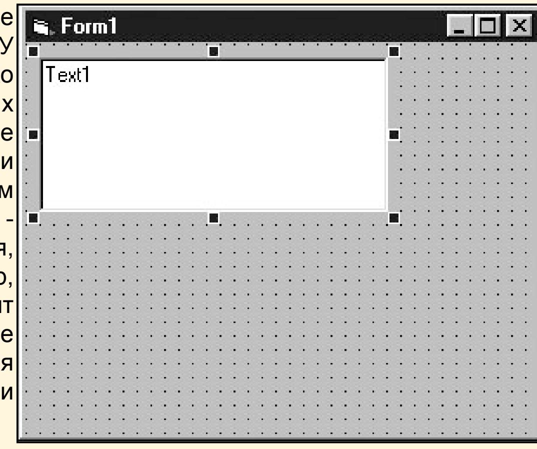
Теперь мы имеем форму - заготовку нашей программы, без которой нам пока не обойтись. Бывают программы, выполняющие определенную работу еще до появления на экране основной формы. У таких программ форма является не основным модулем, а уже, так сказать, вторичным, а роль стартового модуля выполняет стандартный модуль с расширением BAS, содержащий функцию Sub Main(). Но мы оставляем все как есть. Как и зачем это переопределяется мы рассмотрим в последующих выпусках, а пока займемся наполнением нашей основной формы компонентами.

Итак, наша первая форма является основной. Это значит, что именно с ней будет "общаться" легальный пользователь программы MyComPad.exe. А значит, здесь должно находиться все, что наиболее часто оказывается необходимым при работе с программой. Первое: текстовое окно. Ведь текстовый редактор без него немыслим. Второе: функции открытия и сохранения файлов. Третье: что-нибудь такое, чего нет в других подобных программах. Скажите-ка, есть ли смысл создавать точную копию "форточного" Блокнота? Правильно, нет.

В Блокноте есть много пренеудобнейших вещей, которые, к сожалению, уже ничто не исправит… (Вернее, не добавлены многие нужные штучки). Доводилось ли Вам сохранять html-страницу путем открытия ее гипертекстового кода Блокнотом? Вряд ли. Но не это главное: в отличие от WordPad`a Блокнот не знает, что самая главная папка на "винте" Мариванны - "Мои документы", тогда как сохранение той же страницы через Нотопад происходит окольными путями - через кэш, а это - мучительное пере…ное время онлайн. Кроме того, отсутствует список MRU (Most Recently Used). Естественно, чем больше мы нагрузим на хрупкие плечи Блокнота, тем ниже тот будет корчиться под гнетом непосильных задач. От этого может пострадать скорость. Ну, а Блокнот по определению простейший текстовый редактор.

Итак, щелкаем в панели элементов управления на TextBox. В Visual C++ оно почему-то зовется "Edit", а в VBA-редакторе локализованного MS Word 8.0 (Alt+F11) - просто "полем". Подводя курсор мыши поочередно к каждому ЭУ, можно увидеть подсказки, которые и укажут на его имя. Думаю, не ошибетесь.

Можно растянуть поле ввода на определенное значение по ширине и высоте, если главное окно будет фиксированных габаритов, и, соответственно, текстовое поле тоже. Но наш редактор будет обладать свойством адаптации к новым размерам окна в ходе выполнения программы. Такая возможность существует благодаря событию Resize любого окна в ОС Windows. Естественно, функция, обеспечивающая такую возможность, называется Form_Resize. В ней-то мы и напишем необходимый код для реакции "поля" на ресайзы формы, поэтому о его размерах пока беспокоиться не нужно. Собственно, сейчас форма должна принять примерно такой вид:

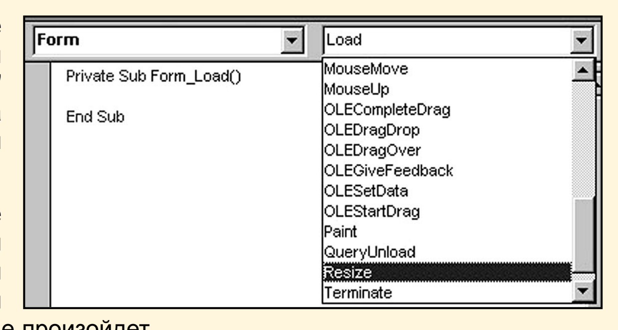


Теперь немного вспомним о свойствах. Наиболее важными сейчас для нас и нашего текстового ЭУ являются следующие: имя ЭУ, так как нам частенько придется набирать его с клавиатуры (в целях самоистязания назовите текстовое поле TheFirstTextBoxInMyFirstTextEditor, или TextBoxForUserTypedTextInputAndSaving). Я же таковым себя не считаю и называю ЭУ либо аббревиатурами, - не так много в программе форм, чтобы в них запутаться, либо одним каким-либо значущим словцом. Например, либо txtMTB, либо txtMain. В обоих случаях txt говорит сам за себя, а Main скажет нам о том, что это - главное поле программы. И даже спустя год после создания проекта Вы с легкостью вспомните, что это такое и зачем оно Вам.

Кстати, о суффиксах. Известный следопыт в области исследования производительности различных встроенных в Visual Basic функций и даже отдельно взятых операторов (!) Брюс МакКинни, - его книга "Hardcore VB" популярна во всем мире, "юзающем" ВБ, - отвел целый ее раздел на доказательство того, что общепринятое именование в среде разработки Visual Basic просто необходимо. Мало того, венгерский стиль, который представляет и мой пример именования, оптимален и рекомендуется для использования в качестве стандарта. Скажу больше: где-то в Сети Интернет я набрел на сайт о "Common Coding Conventions". Не посмотрев на объем скачанного файла, я запустил его на печать, но вскоре отменил ее - уж больно велик документец.

Бейсик же предпочитает именовать компоненты интерфейса по-своему: TextBox1, Label1. Я - не МакКинни. Иногда (когда у меня ОДНО поле) я на это просто не обращаю внимания.

Для того, чтобы пользователь программы мог вводить текст абзацами а не единой строкой, поле должно понимать нажатие на клавишу "Ввод" как текстовый абзац. Для этого в палитре свойств (если она еще не видна на экране, нажмите F4) для текстового поля [как-вы-там-его-назвали] свойство Multiline изменим на True. (Перечислимые войства, в том числе Boolean, переключаются двойными щелчками циклически).



Теперь необходимо обеспечить растягивание поля соответственно формату окна. Дважды щелкнем на форме. "Умолчательным событием" для нее является Form_Load. Из правого списка доступных событий, расположенного на панели объектов/процедур, выбираем Resize:

Примечательно вот что: после первого же прогона кода (F5) неиспользованные заготовки событий, функций и процедур автоматически удаляются. Если же поставить хотя бы один значок комментария (') в пустом шаблоне, этого не произойдет.

Бейсик приготовил нам заготовку для события Resize формы. Не перемещая никуда курсора, вводим текст, после чего имеем законченный код для деформации формы:

```
Private Sub Form_Resize()
  Text1.Left = 0
  Text1.Top = 0
  Text1.Width = Form1.Width
  Text1.Height = Form1.Height
End Sub
```

Код выглядит хоть и синтаксически безукоризненным, зато с точки зрения профессионализма - никудышним. Забегая вперед, замечу: в тех случаях, когда в коде речь идет о той форме, которая и содержит данный код, ее имя можно заменять оператором Me. Переводить не нужно?

Вариант того же кода:

```
Private Sub Form_Resize()
  With Text1
  .Left = 0
  .Top = 0
  .Width = Me.Width
  .Height = Me.Height
  End With
End Sub
```

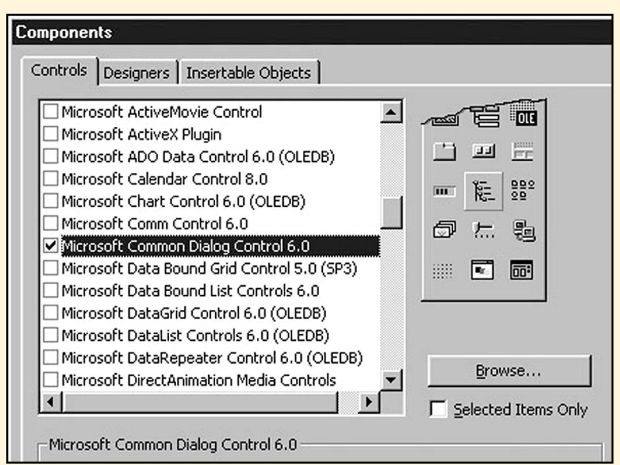
Чем это лучше? Во-первых, теперь нет зависимости от имени формы. Если длина кода достигает пяти-шести экранов, визуально выловить имя давно переименованной формы бывает нелегко. Во-вторых, здесь только две буквы.

Да! Уже можно проверить прогу в работе! F5 - и она запущена. Попробуйте набрать текст. Все происходит по законам Windows. Доходя до края поля, текст прыгает на следующую строку. Нажимая абзац, Вы форсируете переход на новую строку и переводите каретку в начало.

На заметку: сам по себе абзац - не что иное как унаследованные из телеграфа скомбинированные операции: переход на новую строку (в Ворде это Shift+Enter) и возврат каретки. В нашем случае каретка - курсор. Оба действия можно выразить и общепринятыми константами Бейсика: vbLf и vbCr. Каждой из констант соответствует определенное значение. Сумма их равна 13. Это и есть ASCII-код абзаца (наберите в MS Word). Более "запоминаемый" эквивалент - vbCrLf.

Все отлично получается, однако набранный текст невозможно сохранить!

На помощь придет элемент управления CommonDialogs. Точнее сказать, группа элементов, объединенных в один ActiveX-контрол. Нажав правой кнопкой мыши на панели компонентов, находящейся справа, а затем выбрав меню Components…, увидим список инсталлированных в Систему контролов, которые уже сейчас можно использовать "в полный рост". Однако пока не стоит отвлекаться на многообещающие надписи в списке: мы ищем Microsoft Common Dialog Controls 6.0.



Чтобы добавить его в форму, неплохо бы вернуться в режим дизайна (разработки интерфейса) через Project Explorer (Ctrl+R). Двойным щелчком в упорядоченной древовидной иерархии программных объектов можно легко доступиться к желаемой форме.

Броузер проекта вверху своего окошка имеет также кнопки для разных режимов: разработки интерфейса (окна, кнопки, прочая чепуха) и кода. В том же Броузере проекта нажатие правой кнопкой мыши на объекте приведет к появлению меню, в котором имеются два полезных на данном этапе подменю: View Object и View Code.

ЭУ MS Common Dialogs является невидимым компонентом во время выполнения программы. Это и неудивительно. Было бы странным появление на экране, скажем, таймера. Посему не затрудним себя его расположением на форме, а размер нам и подавно не удастся изменить, и просто дважды кликнем прямо на его пиктограммке в панельке компонентов. Бейсик закинет его на серединку формы - он так поступает со всеми Контролами - имейте ввиду...

Пока не забылось, переименовываем его с CommonDialog1 на CD, к примеру.

Общие (стандартные) Диалоги "Форточек" предоставляют нам стандартные блага, а именно те диалоговые окошки, которые так уже приелись в ОС Виндоуз, - это и "Сохранить/Сохранить Как…", и "Открыть", и "Выбор принтера", и даже "Выбор цвета". Большинство программ, написанных на языках, поддерживающих ActiveX-технологию, используют эти диалоги. Однако эти же диалоги вызываются путем использования встроенных в Систему функций API. Но об этом - рановато. На данном этапе мы используем готовый компонент.

В палитре свойств найдем пункт Custom… Нажмем и увидим диалоговое окно Property Pages.

Очень многие компоненты позволяют задавать их свойства через вот такие страницы свойств.

DialogTitle - надпись в заголовке окна

FileName - при сохранении - имя файла по умолчанию, в случае же с открытием файла - получаем динамически.

InitDir - в момент открытия диалог отображает содержимое какой-то исходной директории. Указанное в таблице значение отобразит корень диска с.

Filter - т.н. маска, по которой отфильтруются только указанные файлы, в нашем случае только текстовые файлы. Кроме того, в строке "Тип файлов" диалога будет дописано описание типа.

Остальные свойства оставим со значениями по умолчанию.

Не забудем поставить галочку CancelError: в случае, если пользователь нажмет кнопку "Отмена", произойдет ошибка. Но мы-торасполагаем мощнейшими средствами перехвата таких мелочей с помощью перехода кода. Но для начала - создадим меню: мы ведь должны как-то сказать программе о нашем желании сохранить текст...

На самой верхней панели Среды Разработки найдем пиктограммку с изображением меню. Ее трудно не найти, признаюсь, надо обладать талантом.

Совет:

Если после закрытия окна страницы свойств кнопка редактора меню осталась неактивной, просто щелкните на форме.

Работа с Редактором меню, в принципе, заключается в следующем.

По сути, есть три вида меню: верхнего уровня, ниспадающие, и, собственно, функциональные подменю, нажатие на которых приводит к исполнению каких-либо ассоциированных участков кода, функций или процедур.

К первым относятся такие меню, как File, Edit, View и т.д. Пользователь использует такие меню, чтобы получить либо ниспадающее функциональное меню, либо промежуточное, которое может содержать ответвтвления, ведущие ко вторым. Справа от таких всегда находится характерный треугольничек.

На самом деле все предельно просто, если только Вы не сегодня освоили Windows.

Редактор меню воспроизводит эту иерархию несколько упрощенно, и поначалу немного неудобно, однако по прошествии времени легко убедиться, что здесь все на своем месте и большего не надо.

Во-первых, необходимо построить Верхние меню (Top Level Menus). Сразу создадим меню Файл, Правка, Вид, Справка, поочередно вводя информацию в поля для Надписи меню (Caption) и Имени меню (Name). Если Вы что-то недопишите, Бейсик грязно выругается и не даст Вам больше никогда в этой жизни закрыть Редактор Меню.

Нажимая кнопку Next, Вы заставляете IDE перейти к следующему пункту меню, при этом как бы подразумевается, что Вы ничего не забыли. Однако всегда можно вернуться к любому пункту, щелкнув на его строке.

Если необходимо расставить акселераторы в меню верхнего уровня, как, например, реализовано в большинстве программ для Windows (замечаете подчеркивания под некоторыми из символов в меню?), то рекомендую в тексте меню перед желаемым символом поставить значок амперсанда (&), например, так: &Файл. Теперь пользователь нажимает это меню с клавиатуры: Alt+Ф. Это относится ко всем компонентам, имеющим свойство Caption.

Если же необходимо все-таки отобразить амперсанд вместо акселератора, ставят два амперсанда (&Seek&&Destroy… Простите, &Find&&Replace).

Далее также просто. Нажимаем Insert и "отодвигаем" новое меню (укомплектованное как минимум надписью и именем) вправо с помощью кнопки-стрелки, указывающей в том же направлении. Тем самым мы говорм Бейсику: "на один уровень глубже, плз". Если же засунуть его подальше, чем это возможно (вразрез законам логики и физики), опять посыпится брань: "Menu Item Skipped A Level" - Элемент Меню Проскочил Уровень!

Возможны, наверное, и другие казусы в меню, пока не появятся навыки в работе с Редактором. Не стоит забывать, что навигация производится кнопками со стрелками: вверх, вниз, вправо, влево. Ну что еще сказать?

Акселераторы. Редактор меню предлагает целый набор комбинаций, перечисленный в выпадающем списке Shortcut.

Логично было бы назначать комбинации согласно неким привычным стандартам де-факто. (За эталон можно взять тот же MS Word).



Если Вы все правильно выполнили, Редактор Меню закроет окно и на форме окажется новоиспеченное меню, причем высота главного окна изменится с учетом высоты самой формы-контейнера. Теперь можно нажимать на конечных меню нижнего уровня, при этом Бейсик будет содействовать Вам, создавая шаблоны событий Кликов по менюшкам. А Вам останется только вписывать туда по парочке строчек кода.

Например, чтобы вызвать диалог сохранения документа, просто добавим код для меню mnuSave.

Не следует, вообще-то, забывать: в MS работают люди с юмором. Наверное, поэтому диалоговые окна, за которые мы тут костьми легли, на самом-то деле ничего и не делают. То же сохранение документа возложится на нас, открытие файла - тоже, печать просто так не произойдет, да и цвет не назначится "автоматом" - всюду необходимо прописать капельку кода. На самом деле диалоги - это функции, "вшитые" в ОС и предназначенные для ролучения какой-либо конкретной информации: имя файла для открытия/сохранения, цвета (его шестнадцатеричного эквивалента) и так далее.

Существует наглядное этому доказательство:

```
Private Sub mnuSave_Click()
  On Error GoTo Metka1 ' При ОТМЕНЕ
                  ' переходим в тихое местечко
  CD.ShowSave
  Text1 = CD.FileName
  Exit Sub
  ' --- по идее, тут-то все и завершилось ----
  '
  Metka1:
  MsgBox "Юзер нажал Отмену..." & vbCrLf & _
  "Здесь у нас есть выбор: либо выходить по-хорошему," _
  & vbCrLf & _
  "либо завершить работу аварийно", _
  vbOKOnly + vbCritical, _
  "А это - мой заголовочек"
  Exit Sub
End Sub
```

Теперь текстовое поле содержит имя сохраняемого файла - ни больше, ни меньше. Впрочем, контрол ведь не зря называется всего лишь "Диалогами"…

В предыдущих уроках мы поместили в основную форму MyComPad`a стандартные диалоги при помощи управляющего элемента CommonDialogs, который переименовали в CD (от Common Dialog). Единственная задача этих диалогов - получить нужную информацию, в то время как ни сохранение файла, ни его открытие, ни печать, ни другие действия в отношении файловой системы и ОС автоматически не происходят. ЭУ выступает в роли оболочки, в которую помещены сложные вызовы функций API Windows. Нас, как начинающих программеров, это пока вполне устраивает.

Итак, решено связать диалоги открытия и сохранения файла с нажатием пункта меню, которые мы создали в Редакторе Меню (Menu Editor). Если конечному пункту меню (т.е. меню нижнего уровня) еще не "назначено" никакого кода, при нажатии его в режиме разработки IDE Visual Basic приготовит стандартный для него шаблон-функцию.

В этом случае при нажатии на этот пункт меню Бейсик напишет в окне кода:

```
Private Sub mnuSave_Click()
  '
End Sub
```

Это - стандартная функция-обработчик события "Клик" для нормального пункта меню. Пустые скобки означают, что мы не в силах передать этой функции никаких значений (без применения специальных трюков, рассматриваемых далее в цикле публикаций).

Из этого следует, что для вызова необходимой нам встроенной в CommonDialog1 функции нам не нужно передавать никаких аргументов.

Как же происходит собственно запись в обычный текстовый файл?

Существует несколько способ открытия и оперирования содержимым файла, записью его изменений, формата данных, записываемых в этот файл. Наша цель - создание программы типа Блокнот, управляющая чисто текстовой информацией, во всяком случае, данные, преподносимые ею, будут представлены в виде чистого текста. Для таких целей нам понадобится лишь последовательный способ доступ к файлу.

Как открытие для чтения, так и открытие для записи используют оператор Open, за которым следует, и это естественно, имя файла, который программа пытается открыть, в кавычках-"лапках", потому что имя файла имеет тип данных Строка (String), а в VB все строковые данные (не переменные типа Строка) указываются именно в таких кавычках. Например:

```
MsgBox "Сегодня " & Format(Date, "dd.mm.yyyy").
```

Последовательный доступ к файлу

Рассмотрим принципы сохранения текстового файла на винчестере максимально подробно, так как подобные вещи - фундамент программирования вообще, и в процессе написания программ довольно часто приходится записывать информацию на жесткий диск а также считывать с него.

Итак, приступим. Первое, без чего нам не удастся начать - имя файла. Для получения оного используем ОбщиеДиалоги (CD). Сразу определимся в значимости обсуждаемого кода: чтение/запись являются, в общем-то, часто используемыми рутинными задачами, поэтому резонно вынести их в отдельную функцию. Если так, в дальнейшем мы сократим себе кучу времени, используя ее вызовы типа этого:

```
Dim bResult As Boolean
bResult = SaveFile("c:\MyTextFile.txt")
```

Если записи в файл не случится, bResult примет значение False, наоборот - True.

Не правда ли, все предельно просто? Да, но сначала нам придется поработать, создав эту самую функцию.

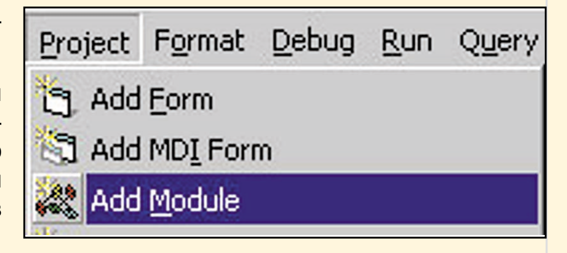
Вопрос: Почему функцию, а не процедуру?

Ответ: По многим причинам запись в файл может не произойти - отсутствует диск в флоповоде, нехватка места на диске, другие форс-мажорные обстоятельства… В таких случаях рекомендуется хоть как-то отслеживать результаты. Другими словами, получать от нее значение. Предлагаю все-таки создать функцию, получающую значение "Да", если все прошло без затруднений, и "Нет" - в противном случае. Кроме этого доступ к функции можно обеспечить и для всех форм проекта, его модулей - сделаем-ка ее публичной и поместим в обычный модуль.

Добавление "стандартного модуля" к проекту

Поместив функцию в "стандартный" модуль, ее можно будет использовать многократно в других проектах, просто добавив в новый проект программы существующий модуль с расширением bas.

Для этого IDE имеет меню Project>Add Module



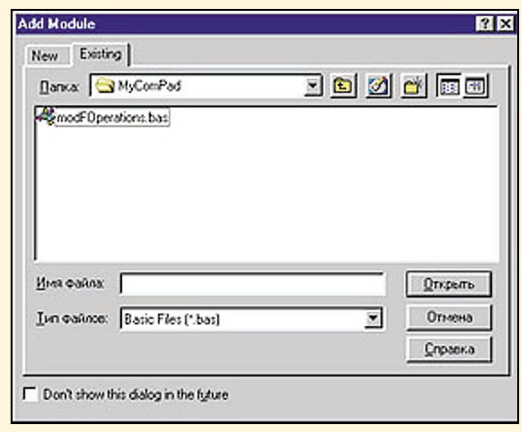
и вкладку в его диалоге - "Existing"

Как только вы добавили модуль в проект, он уже играет свою роль. Конечно, при условии, что функции в нем Public.

Добавление функции

Для того, чтобы создать свою первую функцию, потрудитесь проследовать в меню Tools>Add Procedure. Дайте ей уникальное имя латиницей, и запечатлите это в диалоговом окне - все очень просто и наглядно. Флажок All local variables as Static можно перевести как Все локальные переменные - статические.

Это значит, что переменная, объявленная внутри этой функции, сохранит свое значение до следующего вызова. Бывает, и такое оказывается необходимым. Однако сейчас оставьте флажок неотмеченным.



Ложечка дегтя: Бейсик, каким бы умницей он ни был, ни за что сам не догадается, какого типа аргументы Вы собираетесь выдвигать этому куску кода на обработку. По этой причине в созданном им шаблоне в скобках (и именно там!) впишем FileName As String. (Обращайте внимание на автозаполнение и выпадающие варианты). Имя передаваемой переменной может быть и другим - выберите наиболее удобное для Вас. Однако дальше в тексте мы используем упомянутое выше.

Исходя из вышесказанного, объявление функции должно выглядеть примерно так:

```
Public Function SaveFile(FileName As String) As Boolean
'
End Function
```

В итоге мы имеем локальный обработчик события нажатия меню "Сохранить" mnuSave_Click в модуле формы Form1 и SaveFile в Module1.

(Его можно переименовать точно так же, как элементы управления, имя "стандартных" модулей не играет ключевой роли. Требования именования модулей такие же, как и в отношении ЭУ).

Так как SaveFile является многократно используемым кодом, в процедуре клика по меню пропишем в нем не сам код записи, а обращение-ссылочку, если можно так выразиться.

У Вас должно получиться вот что:

```
Private Sub mnuSave_Click()
  Dim bResult As Boolean
  bResult = SaveFile(CD.FileName)
End Sub
```

Теперь - самое важное. Необходимо наполнить пустышку-функцию начинкой. Переходим в модуль через Project Explorer (Ctrl+R).

Отыскиваем созданный шаблон функции.

Функция SaveFile

Здесь Вам необходимо напомнить об областях видимости: модуль не в курсе об элементах управления, находящихся вне его. Поэтому нам не избежать указания формы-контейнера контролов.

Итак, "гвоздь программы" здесь - CD. Мы ждем от него имени файла. Поэтому:

```
Form1.CD.ShowSave
```

Однако ошибки (даже не критические) рекомендуется отлавливать еще на стадии зарождения: добавляем пустую строку перед набитым только что кодом и пишем следующее:

```
On Error GoTo Metka1 ' При ОТМЕНЕ
'переходим в тихое местечко.
```

Вы спросите, какая связь между ОТМЕНОЙ и ОШИБКОЙ? Загляните в свойство CancelError

ОбщегоДиалога. Это - всего лишь своеобразная реализация перехвата Cancel`a. Если установить значение False - прога будет вылетать со свистом при каждом Esc.

Получив имя файла от CD, мы подошли собственно к открытию и записи в него.

Последовательный способ открытия файла (именно такой способ используется при работе с обычным текстом, например, Блокнотом) состоит из нескольких простых операций.

Сначала некая переменная-счетчик принимает значение типа Integer для строгого учета открытых в данный момент файлов, затем ею выбирается нужный (благодаря этим же идентифицирующим счетчикам) и происходит непосредственно запись - оператором Print (печать).

Однако это вовсе не значит, что Вы обязаны считать все открываемые Вами файлы. FreeFile - лучшее средство в таких случаях. Присвоив ЭТО переменной типа Integer, вы автоматически получаете СВОБОДНЫЙ номер-идентификатор.

Идем далее. Все, что нам нужно, это [Открыть CD.полученное-имя-файла для-вывода в-качестве #Свободного-файла]. На практике так оно и есть:

```
Open CD.FileName For Output As #FileNum
```

Хотелось бы обратить внимание на то, что довольно часты ситуации, когда из-за путаницы в коде открытый файл так и не закрывается программным способом, а при попытке повторного его открытия возникает ошибка Access Error (ошибка доступа).

Совет:

Как и в случае с If…Else…End If лучше набить болванку сразу, отбив абзацами места для строк кода - в таком случае Вы избавляетесь от длительных поисков багов по невнимательности.

Для того, чтобы определить, изменен ли файл, то есть, есть ли смысл писать изменения на диск, необходимо создать переменную типа Boolean (True/False). Если мы назовем ее Dirty, то каждый раз при нажатии клавиш в текстовом поле txtMain Формы1 Dirty будет принимать значение True, а каждый раз при сохранении файла/ревертировании Dirty будет принимать значение False. Так как важность этой переменной неоспорима, мы выделим ей место в "стандартном" модуле, сразу под Option Explicit. Теперь ее видно "за километр" из любого уголка MyComPad`a.

Option Explicit - злорадная штуковина, которая хихикает, когда Ваша переменная не объявлена.

Зачем? В принципе, можно обойтись и без Option Explicit, но тогда в случае опечатки будет создана еще одна, неправильная переменная-призрак типа Вариант, (съедающая, к тому же, оччень много системных ресурсов), но не это главное! Вы - ни слухом, ни нюхом а "правильная" переменная остается не у дел.

Смысл происходящего можно изобразить на пальцах примерно так:

```
[Печатаем в#идентифицированный-файл, содержимое-текстового-поля]
```

В контексте Бейсика функция должна выглядеть подобно этой:

```
Option Explicit
Public Dirty As Boolean

Public Function SaveFile(FileName As String) As Boolean
If Dirty Then
    If Form1.CD.FileName = "" And FileName = "" Then
         Form1.CD. DialogTitle= "Сохранить Как"
         On Error GoTo Metka2
         ' При ОТМЕНЕ переходим в тихое местечко Metka2
         Form1.CD.ShowSave
         GoTo Metka1
    Else
          Form1.CD. DialogTitle= "Сохранить"

    Metka1:
          Dim FileNum As Integer
          FileNum = FreeFile
          Open Form1.CD.FileName For Output As #FileNum
          Print #FileNum, Form1.Text1.Text
          Close #FileNum
          Dirty = False ' Файл сохранен!
    End If
Else
     MsgBox "Файл не нуждается в сохранении", _
     vbOkOnly+vbExclamation, "MyComPad 1.0"
End If
    SaveFile = True
    Exit Function
    ' ------- по идее, тут-то все и завершилось ----------
    '
    Metka2:
    Beep ' Характерный Бип
    MsgBox "Юзер нажал Отмену..." & vbCrLf & _
    "Здесь у нас есть выбор: либо выходить по-хорошему," _
    & vbCrLf & _
    "либо завершить работу аварийно", _
    vbOKOnly + vbCritical, "А это - мой заголовочек"
    Dirty = True ' Файл по-прежнему не сохранен
    SaveFile = False
    Exit Function
End Function
```

Функции можно имя файла и не передавать. Проверка

```
If Form1.CD.FileName = "" And FileName = "" Then
```

установит факт пустой строки - в таком случае вызов функции будет таким:

```
MyVar = SaveFile("")
```

и потребует ее от пользователя через CD.

Переменные типа Boolean имеют одну особенность. Вполне законным и справедливым считается и такая запись: If Dirty Then … Однако можно написать и так: If Dirty = True Then… И наоборот, проверка отрицательного значения: If Not Dirty Then и If Dirty = False Then абсолютно идентичны.Выбирайте, что Вам больше по душе.

Как внимательнейший читатель, возможно, заметил, в отличие от второй метки, первой не предшествует Exit Function. Причина в том, что код под первой должен выполниться в любом случае, а под второй меткой у нас находится "аварийный выход". Играя метками, можно добиться эффективного кода, хотя подобным образом эффективность может выразиться лишь в лаконичности, краткости. Однако следует остерегаться "подводных камней" в переходах GoTo, Exit…. Например, открыв файл, Вы должны обеспечить и его закрытие, но если программа "перешла" куда-то, забыв о закрытии, файл окажется заблокированным (Ошибка доступа). Поэтому никогда не лишней будет проверка всех возможных вариантов в системе If…Else…End If а также внимательное изучение ситуаций переходов. Ведь лучше потратить пару минут на такую мелочь, чем выгребать потом по-взрослому J, когда код

программы достигает мегабайта.

Итак, пример показывает две системы If…Else…End If, причем одна оказывается внутри другой. Если суждение Form1.Dirty является Истиной (а-ля Ворд), другими словами, файл изменен, то проверке подвергается наличие Form1.CD.FileName. Если такового нет, значит, файл еще не сохранялся - следует показать юзеру окно диалога Save, причем его заголовок станет "Сохранить Как". После этого, если не наступила ошибка "Кэнсэл", идем к метке1, а именно приступаем к записи.

Если же Form1.Dirty не соответствует реальному положению дел, можно выдать пользователю сообщение MsgBox "Файл уже сохранен", или что-то типа этого, а можно нацарапать Exit Function. Это даже логичнее, так как не стоит вообще заострять на этом внимание оператора ЭВМ. Большинство солидных программ после сохранения вообще делают кнопки и меню сохранения недоступными - не кликнешь! Кстати, свойство Enabled заключается не только в изменении внешнего вида ЭУ, - то самое событие Клик просто не наступает. Так что есть смысл после успешной реализации функции устанавливать значение свойства Enabled меню mnuSave в False, и наоборот: при отмене, ошибке и изменении текста в поле - в True, тогда юзер периодически сможет кликать по менюшке.

Еще одна особенность листинга. В любом случае, независимо от результата функция принимает значение - так, для прикола (SaveFile = True/False). Например, где-то в основной форме понадобится узнать сумел ли пользователь сохранить файл. Вспомните, как ведет себя Блокнот по нажатию кнопки с крестиком контролбоксе формы (Выход) когда набрано пару символов в его основной рабочей области. Выход - только через диалог "Сохранять/Не сохранять".

Лучшая проверка программ, написанных на Visual Basic - полная компиляция, когда, весь исходный код прогоняется через компилятор. В случае малейшей неприятности процесс компилирования будет остановлен, курсор установлен в то место, где допущена ошибка, номер ошибки (Вам это ни к чему) и ее описание будут объявлены. Кстати, об ошибках. В наиболее узких местах в режиме предварительного проектирования программы полезно расставить КробкиСообщений, несущие информацию типа Err.Code или, если Вы ленитесь выучить наизусть каких-то пару тысяч кодов "Форточных" ошибок, хотя бы Err.Description - естественно перед Exit Function/Exit Sub.

Состояние документа: Dirty или Not Dirty?

Теперь надо подумать о переменной Dirty. В каких случаях изменяется содержимое текстового поля txtMain? Верно: нажатие клавиши, вставка из буфера и так далее. Все это изменяет ТекстБокс. К счастью, нам не придется перечислять все клавиши, нажатие которых приведет к изменениям в поле - для ЭУ TextBox предусмотрено одно удобное, общее для всех ситуаций событие Change, которое все учтет. Поэтому при загрузке формы можно заблокировать меню mnuSave таким образом:

```
mnuSave.Enabled = Dirty = False
```

так как сохранение пустого файла смысла особого не несет, а в txtMain_Change() прописать

```
mnuSave.Enabled = Dirty = True
```

Если мы в режиме разработки не назначили свойству FileName контрола CommonDialogs никакого значения, программа будет показывать диалог "Сохранить Как", как мы того и хотели. В противном случае запись файла будет происходить незаметно для пользователя.

Ну, и перед более насыщенной кодом темой "Циклы For…Next. Коллекции", в которой будет рассмотрен конкретный пример заполнения элемента управления ComboBox инсталлированными в Систему шрифтами, замечу: благодаря функции SaveFile можно контролировать состояние файла: сохранен он или нет, и вообще, сохранялся ли когда-либо. Например, в событие Form_Load можно поместить такой код:

```
If Not Dirty Then
  End ' Завершение программы
  Else
  Dim bResult As Boolean
  bResult = SaveFile("CD.FileName")
  Select Case bResult
  Case True
  End
  Case Else
  Beep
  MsgBox "Произошла ошибка: " & vbCrLf & _
  Err.Description, vbOkOnly + vbCritical, "Error!"
  Exit Sub ' Не дать пользователю потерять текст
  End Select
End If
```

Итак, мы дали пользователю возможность сохранять файлы. (Негодование публики: сначала нужно что-то открыть, чтобы его сохранить. Ну, это из области теории "яйцо-курица").

Действительно, впору заняться созданием функции открытия файлов, затем - дальнейшим совершенствованием элементов интерфейса и их взаимодействием между собой.

Как читатель, наверное, помнит, мы договорились, что функциональные вопросы, касающиеся чтения/записи в файл можно считать неизбежностью в программировании вообще. Поэтому лучше создать функции, обращаясь к которым, можно будет писать или читать файлы практически не задумываясь о механизме их реализации.

(Хотя на данном этапе упрощение - не Вам на руку. Практика сложных приемов - лучшее средство хорошенько закрепить материал). Для тех, кто уже программирует на каком-либо языке высокого уровня (VC++, Delphi), сия тема покажется скучной, тем паче, что Ваш покорный слуга имеет целью растолковать сущность Visual Basic даже самым-самым "тугим". Отсюда и тягучее толкование прописных истин. Одним словом, я подвожу читателя к "азам" Бейсика. Асы, Маги, Гуру и Шаманы Visual Basic могут перекурить...

Ну-с продолжим. Запись текстового файла производится ключевым словом Print, причем не просто так, а после определенной подготовки к записи. Запамятовали? Тогда читайте внимательно.

Для начала программа должна получить идентификатор типа ЦелоеЧисло (например, 1, 2, 3, 67, 233....), по которому можно обращаться к открытым ею файлам, например, таким образом:

[Эй там, а ну-ка записать в четвертый строку "Ты выписал "Мой Компьютер"?", в восьмой - цифру 2, а третий - ваще закрыть!]. Пример нелепый, но примерно так для пользователя должен представляться макет оперирования файлами.

Реальный код выглядел бы примерно так:

```
Print #4, "Ты выписал "Мой Компьютер"?"
Print #8, "2"
Close #3
```

Вопрос: где на практике может понадобиться фиксированная нумерация файлов?

Ответ: в программе, использующей в своей работе только определенное количество файлов. Ни больше, ни меньше. К примеру, ее цель - суммирование каких-либо чисел, находящихся в разных файлах. Причем количество слагаемых постоянно и известно заранее. Разработчик в курсе дела. Он всегда помнит, что файл #3 - это c:\Work\Accounts\Member_003.txt, #1 и #2 нужно скачать по сетке с такой-то машины, а файл-репорт #4 - это c:\Work\Accounts\Result.txt.

Как ни крути, а файлы остаются при своих неизменных идентификаторах. Не буду мучить Вас аллегориями, Бейсик намного лаконичнее и последовательнее меня. Он требует от Вас такого синтаксиса:

```
Dim iMyVar As Integer
IMyVar = FreeFile
Open "c:\Test.txt" for Output As #iMyVar
Print #iMyVar, "Строка текста," & _
    vbCrLf & "и еще одна!"
Close #iMyVar
```

Конечно, если Ваше изучение английского языка закончилось на зубрежке алфавита в школе, - мои соболезнования, Вы изучаете не тот язык…

Однако даже вода камень точит - словарь в руки, и - ...

Вот так организована запись текста на винт. А как же устроено чтение с него?

Запомни, уважаемый читатель, Бейсик - не язык программирования. Почему? А потому, что уж очень он прост. У него есть еще "недостатки", о которых я расскажу позже. Кстати, покажу, как ими эффективно пользоваться. Здесь отлично работает теория "от обратного": замените Output на Input, а Print на Line Input (внимание: пробел!), - сложно? Нет. Однако это - в идеале. На практике "непрограммисты" на Бейсике пишут где-то так:

```
Public Function ReadFile(strFileName As String, _
                     TextBoxControl As TextBox) As Boolean
  ReadFile = False ' Результата пока нет
  If strFileName <> "" Then
  Dim strTemp As String
  Dim strAllText As String
  Dim iFileNum As Integer

iFileNum = FreeFile
  On Error GoTo Metka3
  Open strFileName For Input As #iFileNum

  While Not EOF(iFileNum)
  Line Input #iFileNum, strTemp
  If strAllText <> "" Then
     strAllText = strAllText & vbCrLf & strTemp
  Else
     strAllText = strTemp
  End If
  Wend

  TextBoxControl.Text = strAllText
  ReadFile = True
  MsgBox strFileName & " открыт!", vbInformation + vbOKOnly, "MyComPad 1.0"
  Close #iFileNum
Else
End If

  Exit Function 'Счастливый конец

  Metka3:
  Beep
  MsgBox "Сорри, конечно, но что открывать-то?" & _
     vbCrLf & "Попробуйте еще раз позже...", _
  vbCritical + vbOKOnly, "Обшибочка, однако!"
  ReadFile = False
  Exit Function

End Function
```

Думаю, стоит объяснить природу процесса построчного чтения подробнее.

Даже опытные программеры частенько убивают зря время в поисках ошибки, а она рядом (я всегда склонен считать, что все "ошибки компьютера" - ошибки по невнимательности оператора). Но, отыскав, недоумевают - ужас как глупо и нелепо.

Переменные типа Boolean (это те, которые "либо-либо" - Да/Нет) по умолчанию (это когда Вы умолчали факт создания такой переменной; создали себе, и в ус не дуете) рождаются со значением "Нет". В этом примере летальным исходом не закончилось бы, но, к примеру, (чисто теоретически) функция зовется не ReadFile, а ErrorOpen, тогда при успешном чтении она должна была бы вернуть

противоположное нашему значение - "Нет". Действительно: нет, не было ошибки и файл прочитан - все Окей.

Но вот беда: Вы забыли сделать поправочку. Сначала-то не было никакого результата. Другими словами, при входе в функцию делаем вид, что ошибка уже допущена. Но, открыв файл (к примеру), говорите: "А вот теперь - все нормально. Ошибок нет".

Конечно, ту же функцию можно построить и по-иному, и вообще, можно дважды присваивать значение функции - первый раз при успешном открытии, второй - перед экстремальным выходом, когда файл не найден. Здесь уж фантазия подскажет. Однако я склонен экономить место на диске, поэтому обычно пишу как можно лаконичнее (к статье это не относится).

Поэтому в первой строке кода мы заведомо лжем: ReadFile = False, но по истечении некоторого времени, а именно после непосредственно операции над несчастным файлом, ReadFile становится True, и все вокруг довольны. Естественно, если что - спасительное Exit Function не даст нам этого сделать - программа просто выйдет "ни с чем" - вернее, со значением "Нет". Логично.

Далее следует If strFileName <> "" Then, строка, которая предотвратит использование дальнейшего кода, если strFileName, которую передали в черном ящике функции на обработку, окажется "ничем". Обратите внимание на то, как устроена система If…Else…End If:

```
[Если передано-реальное-имя-файла То
Если ошибка - знаешь что делать (иди на Метку3)
Что-нибудь утворим с этим файлом
Иначе
у нас есть на это достойный ответ - Метка3
Конец Проверки
Дело-сделано-можно-уходить
Метка3:
Результат - Нет
Сообщение об ошибке и
выход]
```

В принципе, так можно схематически описать то, что я наваял в листинге.

"Экстремальный код" как бы живет отдельно от остального кода. Государственной границей здесь служит Exit Function (первый). При отсутствии ошибки дело до второго дойти не должно.

Очевидно, что лексемы Output и Print говорят сами за себя.

Построчное чтение файла

Цикл *[Пока Не Достигнут-конец-файла … Конец цикла]* когда-нибудь доберется до конца файла при помощи **Line Input** и передаст правление коду, который следует за Wend.

Line Input как раз и подразумевает перебор строк файла (кстати, небесконечного). Если бы там был указан иной оператор, End-Of-File никогда бы не наступил.

В построчном способе чтения текст порциями поступает в "карманчик" (strInput), а оттуда - в общий карман побольше (strAllText), и когда все поутихнет, из этого большого кармана текст вывалится в текстовое поле. Которое, к стати, также передано функции в качестве аргумента, поэтому функцию ReadFile можно использовать не только с текстовым полем txtMain, а с любым другим текстовым полем. Следует заметить, что TextBoxControl имеет тип TextBox, и любой другой тип данных (типа Строки, ЦелогоЧисла (Integer), Boolean…) будет грубо отвергнут. Функция работает только с текстовыми полями. Почему? Всмотритесь в TextBoxControl.Text = strAllText. Ведь не у всех элементов управления есть свойство Text.

Совет 1: *Можем сделать нашу функцию более гибкой. Объявим ЭУ как Control, заменим TextBoxControl на Control, а также избавимся от ограничивающего нас четкого указания свойства таким образом:*

```
Control = strAllText,
```

Теперь мы играем со свойствами по умолчанию, а значит, можно использовать функцию и с Label, и с элементом управления Frame, и с кнопкой, а также с любым другим ЭУ, умолчательное свойство которого использует данные типа String.

С другими ЭУ могут быть ошибки синтаксического характера: например, присвоение значения списку ListBox или ComboBox звучит как AddItem, хотя свойство по умолчанию - строка.

Совет 2:

Для пущей гибкости кода можно вписать еще одну систему, основанную на проверке типа элемента управления:

```
If TypeOf Control Is CheckBox Or _
       TypeOf Control Is Label Or _
       TypeOf Control Is CommandButton Or _
       TypeOf Control Is OptionButton Then
   Control.Caption = strAllText
ElseIf TypeOf Control Is ListBox Then
   Control.AddItem = strAllText
ElseIf TypeOf Control Is TextBox Then
   Control.Text = strAllText
End If
```

Как видно, код "подстраивается" под "причуды" каждого ЭУ при помощи вспомогательного оператора проверки ElseIf (ИлиЖе).

Когда цикл прохода от начала файла до его конца окончен, функция принимает значение-результат, при этом, кстати, она просто "помнит" его до тех пор, пока мы не спросим. Если же мы никогда этого не спросим, она так и умрет с ним, никому ничего важного не сообщив. Мы не зря выбрали в качестве контейнера функцию, а не процедуру - сейчас, возможно, Вы не готовы использовать обработку ошибок, и такое прочее, но вскоре….

Теперь следует закинуть все это в тот же модуль, где уже находится созданная ранее функция SaveFile. Нет смысла хранить однородные функции в разных модулях.

Использование функции. Вызов

Откроем окно формы через Project Explorer, кликнем на меню mnuOpen. Если меню еще не создано - самое время поиграться с редактором меню.

Предположим, меню mnuOpen существует, его свойства Visible и Enabled (Видимое и Доступное) установлены в True. Тогда достаточно одного щелчка мышью в режиме разработки - и появится шаблон

для процедуры "Клик_ПоМенюшкеОткрыть".

Теперь наша задача - грамотно вызвать функцию ReadFile:

```
Dim openResult As Boolean
CD.ShowOpen
If CD.FileName <> "" Then _
     openResult = ReadFile(CD.FileName, _
     txtMain)
```

Заметьте - система *If…Then* может обходиться без логического завершения End If, но только в одном случае: если в ней нет Else (Иначе) и она набита в одну строку.

В тех немногих случаях, когда требуется узнать результат работы функции, это можно проделать через посредника-переменную openResult, то есть ту, которая берет на себя задачи контроля за ходом выполнения функции еще при ее вызове. В листинге она выделена жирным.

Отныне Вы вольны делать с файлами что угодно: хоть читать, хоть писать. Стоит только написать нечто вроде

```
Dim TraLjaLja As Boolean
```

строкой ниже - *TraLjaLja = ReadFile(Text14, Text68)* - и все, шестьдесят восьмое текстовое поле получит внутренности файла, полный правильный путь к которому указан в четырнадцатом текстовом поле. Далее, если глубокоуважаемая миссис Windows глюконула (ну… не без этого), файл кто-то бахнул, или вообще там его никогда не было и быть не могло, можно об этом во весь голос сообщить юзеру:

```
If TraLjaLja = False Then _
    MsgBox "Вау! - что-то глючит твоя машинка…", _
    vbCritical + vbOkOnly, _
    "По-моему, диск не системный"
```

и наоборот:

```
If TraLjaLja = True Then Call CheckSpell
. Понятно, что если
TraLjaLja = "Нет"
```

то проверка правописания, мягко говоря, будет … гм… ни в Красную Армию.

Однако не стоит забывать о переменной Dirty (вспоминайте! Эта переменная - неустанный диспетчер у рубильника, на котором красной краской написано: "Файл сохранен"/ "Файл не сохранен").

Вот тут-то мы и сможем использовать "проверочку на вшивость": If openResult = True Then Dirty = False. Именно True, поскольку только что открытый файл - девственен и чист. Если файл не был открыт - то… ничего не делаем, - можем просто развести руками: MsgBox "Что-что?", vbOkOnly, "Тьфу…" И именно по этой причине не нужно себя утруждать оператором Else.

Копирование и вставка текста

Вообразите себе какой-нибудь навороченный текстовый редактор - с "веерными распальцовками" вроде марокоманд, с автозаполнением, интеллектуальным анализом удобочитаемости в духе мелкомягкого Ворда, шрифтовыми фишками, автоформатом, стилями и таблицами, но без элементарных функций копирования и вставки текста. Правда, жуть?

Все, что нам нужно для воплощения мечты в реальность - два меню, расположенных в одном вышестоящем меню "Правка". Если вы впервые читаете наш "сериал", поясню: редактор меню позволяет создавать и иерархически разделять созданные меню. Меню верхнего уровня, т. е. самые главные, в которых находятся выпадающие (и/или конечные), обозначены в нем без отступов. Остальные - с отступами, в зависимости от уровня "подчиненности". Каждое меню должно иметь как минимум уникальное имя в соответствии с правилами именования в VB, желательно надпись (если только она не назначается динамически), и не противоречить законам физики и аэродинамики. Шутка. Однако вы не в силах назначить акселератор Ctrl+C для меню верхнего уровня, а тем более какой-либо код.

Итак, вы создали меню "Правка" (mnuEdit), в нем - "Копировать" (mnuCopy) и "Вставить" (mnuPaste). Рекомендую не идти против привычек пользователей MS Office и назначить комбинации клавиш соответственно Ctrl+C и Ctrl+V.

Щелкаем на меню "Копировать". Появляется шаблончик для ввода кода. Никуда не щелкаем и начинаем набивать текст:

```
If txtMain.SelLength > 0 Then
   Clipboard.Clear
   Clipboard.SetText txtMain.SelText
End If
```

(Попробуйте-ка закомментировать Clipboard.Clear...)

Рассмотрим листинг.

Как всегда, согласно традиции, привожу примерную транскрипцию кода:

```
[Если ДлинаВыделения БольшеНуля Тогда
Очистить БуферОбмена
ПоместитьВБуферОбмена То-Что-Выделено В-ГлавномПоле
Конец Проверки]
```

Если вы, внемля гласу разума, установили традиционные акселераторы, то и нажатие меню, и клавиатурные комбинации копирования текста уже функциональны и ничем не уступают аналогам, реализованным в MS Office!

Вставка из буфера в "Форточках" реализована несколько сложнее, потому как обязывает сперва проверить тип данных, вставляемых в контейнер. Как вы себе представляете вставку скопированного в Photoshop'е рисунка в текстовый блок?

Однако, наша проверка выразится лишь в отфильтровке того формата, который поддерживает текстовый файл, а именно чистый текст.

Таким образом, пропишем в процедуре клика по mnuPaste:

```
txtMain.SelText = Clipboard.GetText(1)
```

В тех случаях, если буфер обмена содержит не текст, а, например, картинку, вставки из буфера вы не дождетесь, причем обработчик ошибки, я думаю, находится где-то в функции Clipboard.GetText - не напрягайтесь в поисках обхода проблемы несовместимости форматов.

Немного лирики

Как я и обещал, рабочими проектами, иллюстрирующими возможности 32-разрядного Visual Basic`a, в моей статье не станут "Хелло, урод" или Нечто-Вроде-Этого. Во-первых, нет смысла ограничиваться таким примитивом по причине "врожденной" простоты Бейсика, а во-вторых (субъективное мнение автора), проект типа "Привет, Мир!" навевает невыразимую скуку.

Я предлагаю добавить в проект плавающую панель инструментов. И хотя в Plain Text-редакторе

найдется не так много команд, кнопки управления которыми можно было бы разместить на этой панели, это даст читателю некий уровень знаний (а может, и навыков), которые он, не сомневаюсь, с удовольствием применит где-нибудь в своем проекте. Еще раз повторюсь, есть смысл заменить существующий Notepad.exe на MyComPad, обладающий многими преимуществами, в которых вы, уважаемый читатель, будете все больше убеждаться из номера в номер - ведь все описанное в этом цикле продиктовано ежедневными практическими наблюдениями. Когда изо дня в день тебя мучают чужие недоработки, промахи, неудобства, приходится изобретать свое колесо.

Несколько слов об интерфейсе

Хочется вспомнить публикацию о шестой версии знаменитого дизайнерского/оформительского пакета, где мне довелось посмаковать улучшениями в области UI. На протяжении многих лет по воле службы мне приходилось использовать продукт с утра до вечера, лазая по меню либо мышью, либо ломая пальцы в сумасбродных альт-комбинациях, потому что стандартных не предусмотрел разработчик, причем наиболее часто используемыми меню оказываются наиболее глубоко спрятанные. Спрашивается, почему не сделать пару кнопок, упростивших бы работу?

Вы можете себе представить QuarkXPress без Measurements Palette? А "Офис" без кнопок?

Некоторые производители софта дошли до возможности полной перенастройки интерфейса конечным пользователем - меню (CorelDraw!), панели (тот же CorelDraw!, MS Office, Lotus WordPro), комбинации клавиш. Да, чаще всего юзеру облом лезть под капот, однако обычно его отсутствие просто утомляет.

Однажды я наткнулся на толстенькую книжечку об "умном интерфейсе". Более сотни страниц ее были исписаны рекомендациями о дизайне программного обеспечения, эргономичности и удобстве. В общем-то, ничего нового, просто поразил подход... Ведь как правило, человек не просто запускает программу - он садится работать, чаще всего надолго. Чем больше интерфейс "подогнан" под его мерки, тем удобнее тот себя чувствует. Соответственно, результаты...

Добавление "плавающей" панели инструментов

Итак, панель инструментов. Какие можно выдвинуть к ней требования, какими качествами она должна обладать, чтобы:

- быть полезной (ее основная задача),
- быть всегда под рукой,
- не причинять побочных неудобств юзерам?

Мне доступ к командам сквозь бесконечные дебри ниспадающих меню кажется весьма неудобным. Почему в Notepad`е нет Ctrl+S - для меня загадка. Такая же, как и отсутствие кнопки "Сохранить как..." в IE любой версии. Зато есть "Cut"! Хорошо, допустим, вам ежедневно приходится вырезать целые сайты... (гм...), но тогда где "Paste"?

Наша программа будет иметь панель инструментов, которую можно либо вызывать, либо прятать - по желанию народа.

Собственно, панель по своему определению всегда под рукой. Другое дело, когда она вообще всегда под рукой, даже если этого не нужно, и

спрятать ее нельзя. Одним словом, она должна быть мобильной и, по возможности, контекстно-активной.

Панель инструментов, как и любое программное окно, строится на основе существующего шаблона формы и производится путем нажатия меню Add Form. Появившийся диалог предложит вам несколько вариантов окон в уже приготовленным минимальным кодом. Стоит только добавить пару строк туда, пару строк - сюда, и элементарный интерфейс готов к употреблению. Просто добавь "воды".

Для наших целей подойдет вполне функциональная модель стандартной формы. Она зовется просто "Form". Выбрав ее из списка с пиктограммами и нажав OK, получаем заготовку, весьма похожую на ту, что мы видели при добавлении первой, основной формы проекта.

Об окнах

Как и любой элемент управления, Форма имеет свойства, события и методы, однако есть определенные различия, которые мы по ходу дела обсудим.

Для начала изменим имя Формы, заданное Средой Разработки по умолчанию, на frmTools. Следуя наставлениям гениального МакКинни, мы уже сделали определенный шаг к созданию суперпрофессионального продукта, применяя венгерскую модель именования (см. пред. вып.). Наша панель должна быть максимально компактной, поэтому следует удалить ее заголовок - перемещение панели будет возможно путем перетаскивания ее за любую часть. Для этого установим свойство Caption в "", ControlBox - в False, BorderStyle - 4 - FixedToolWindow.

Теперь она не имеет верхней строки заголовка, нет кнопки "Закрыть", она не подвержена деформированию (Resize). Кроме того, она должна быть всегда поверх основного окна MyComPad`a. Этого достигают при помощи определенного способа загрузки формы.

Добавьте меню верхнего уровня "Вид" (mnuView), в нем - "Инструменты" (mnuTools), Ctrl+T. Оба меню видимы и доступны, свойство Checked у обоих установлено в False (галочка снята).

Нажав на mnuTools, получаем заготовку для клика меню.

Пишем следующее:

```
frmTools.Show 0, Me
```

Модальные и немодальные окна

Как известно, интерфейс в ОС Windows интерактивен, то есть на запрос пользователя программа совершает действия, соответствующие конкретной ситуации, и наоборот, иногда программа ждет от оператора определенного решения, от которого зависит дальнейший ход событий. Интерактивность заключается в двустороннем обмене информацией, причем обе стороны активно принимают участие в диалоге. Иногда ход выполнения программы невозможен (или крайне нежелателен, бессмыслен) без принятия решения пользователем. Например, нельзя не сообщить пользователю о том, что в дисководе отсутствует диск и процесс сохранения открытого с дискеты файла невозможен. В таких случаях появляются окна сообщения (Пример: MsgBox) с оперативной информацией. Не закрыв окна сообщения, юзер не сможет продолжить работу. Окна, блокирующие доступ к другим окнам проекта или программы, называют модальными. Любое окно, за исключением основного, можно загрузить как модальное (под загрузкой подразумевается показ на экране, чему предшествует невидимая для пользователя загрузка экземпляра Формы в оперативную память), и как немодальное. Загрузка в обычном, немодальном режиме является установленной по умолчанию.

В Бейсике для запуска форм в этих режимах созданы константы - vbModeless, способ по умолчанию, и vbModal, который используют окна сообщений. Константам соответствуют числовые эквиваленты: 0 (vbModeless) и 1 (vbModal). Сама загрузка окна выражается либо методом **Show**, либо **Load**. Первый предпочтительнее второго, поскольку метод Show выполняет метод Load для Формы, если та еще не загружена, а Load покажет не окно, если оно уже загружено, а сообщение об ошибке типа "Object already loaded".

Метод **Load** будет подробно рассмотрен далее.

В предыдущем уроке наш MyComPad приобрел плавающую панель frmTools. Такое ее специфическое положение обусловлено способом загрузки, а именно немодальностью. Для того, чтобы форма не скрывалась из виду при переходе фокуса к другому окну, необходимо указать параметр Owner (форма-владелец, «хозяин»). Отсюда следует, что код загрузки окна frmTools при нажатии на меню mnuTools должен выглядеть так:

```
Private Sub MnuTools_Click()
    FrmTools.Show 0, Me
End Sub
```

Вероятно, Вы помните, что форма, содержащая данный код, может быть указана как Me. Заменив 0 на единицу, получим модальное окно, не позволяющее переключиться на какой-либо иной объект в этой программе.

Рассмотрим наиболее важные свойства окна frmTools:

Свойство	Значение
Name	FrmTools
BorderStyle	**4**-FixedToolWindow
Caption	Нет
ControlBox	False
Enabled	True
Font	Arial
ShowInTaskbar	False
StartupPosition	**3**-Windows Default
Visible	True
WindowsState	**0**-Normal

О свойстве Имя (Name) писалось в предыдущих публикациях. Напомню только, что существуют определенные требования к именованию ЭУ, форм, модулей. Имена не должны содержать нелатинских символов, не должны начинаться с цифр и спецзнаков, и должны быть длиною не более 256 символов. Вроде все понятно.

BorderStyle

Свойство «Стиль рамки» означает не только ее визуальное оформление - BorderStyle подразумевает, сможет ли пользователь деформировать (растягивать) окно, будет ли окно иметь строку заголовка, и какова будет ее (строки заголовка) высота. Так, выбранный нами стиль означает, что окно сможет иметь строку заголовка, высота этой строки заголовка - в стиле панели инструментов, диалоговых окон и т.д., (то есть немного меньше, чем у обычных рабочих окон программ), окно будет фиксированной ширины и высоты. Другими словами, не Sizable.

В таблице приведены другие возможные значения свойства BorderStyle.

Значение	Описание
0 - None	Рамка и заголовок отсутствуют
1 - Fixed Single	Обычная фиксированная рамка
2 - Sizable	Растягиваемая рамка
3 - Fixed Dialog	Фиксированная рамка. Заголовок - крупный, шрифт в заголовке такой же, как и в обычных окнах. Из кнопок управления окном остается лишь крестовидная «Close»
4 - Fixed ToolWindow	Фиксированная рамка. Заголовок - мелкий, шрифт в заголовке мельче, чем в обычных окнах. Из кнопок управления окном остается лишь крестовидная «Close»
5 - Sizable ToolWindow	То же, но с возможностью изменение размеров окна

Caption

Надпись в строке заголовка - это "главная" надпись окна. Обычно Microsoft Word использует ее для указания имени приложения и файла.

Плавающая панель, как уже было сказано, не имеет такой надписи, и в сумме с отказом от кнопок управления формой обладает своеобразной рамкой.

Перемещение такой формы без заголовка осуществляется только благодаря использованию некоторых системных функций (Windows API). По сути, класс окна (ООП и классы будут рассмотрены позже), использующий заголовок, обращается к той же API-функции. Вообще, в Windows ничего не берется ниоткуда - все «трюки» и ухищрения являются умелым использованием системных ресурсов в лице предоставленных системой функций. Забегая вперед, открою маленький API-секрет: чтобы перемещать окно, не имеющее заголовка, поместим в модуле (создадим новый и назовем его APIs) строки:

```
Public Const LP_HT_CAPTION = 2
Public Const WM_NCLBUTTONDOWN = &HA1
Declare Function ReleaseCapture Lib "user32" () As Long
Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd _
   As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
```

Рассмотрим внимательно, что мы наделали.

Сначала мы объявили две константы для функций, указанных ниже. Практически все вызовы API требуют объявления соответствующих констант, в них нуждаются те библиотеки, к которым Вы посылаете запрос или сообщение.

Ниже следуют два объявления функций: ReleaseCapture и SendMessage. О них будет рассказано в главе об API-интерфейсе, а пока - реализация вызова данных функций.

Процедуру события MouseDown окна frmTools следует привести к указанному виду:

```
Private Sub Form_MouseDown(Button As Integer, _
      Shift As Integer, X As Single, Y As Single)
      Dim rc As Long
      rc = ReleaseCapture
      rc = SendMessage(hwnd, WM_NCLBUTTONDOWN, _
            LP_HT_CAPTION, ByVal 0&)
  End Sub
```

Как видно из листинга, функция принимает во внимание многие факторы нажатия кнопки мыши - Button, Shift, X и Y. Передавая соответствующие значения в качестве аргументов, можно локализовать выполнение функции-обработчика. Например, указав кнопку мыши №2 (правая), мы подразумеваем выполнение MouseDown только в случаях райт-кликов. Таким образом для каждой кнопки мыши можно предусмотреть свой код:

```
Private Sub Form_MouseDown(Button As Integer, _
            Shift As Integer, X As Single, Y As Single)
      If Button = 2 Then
         Dim rc As Long
         rc = ReleaseCapture
         rc = SendMessage(hwnd, WM_NCLBUTTONDOWN, _
                LP_HT_CAPTION, ByVal 0&)
      Else 'Не-правая кнопка (бывает и три кнопки)
            MsgBox "Вы нажали кнопку " & Button
      End If
  End Sub
```

Вопрос: Зачем объявлена переменная rc?

Ответ: rc реализована для проверки выполнения поставленной задачи (см. предыдущие публикации). В случае ошибки возвращается значение 1.

Тема аргумента Shift в MouseDown и KeyDown будет еще рассмотрена более внимательно в текущем проекте.

Enabled

Иногда форму нужно оградить от воздействия пользователя, например, при выполнении поиска в длинном тексте, когда вырезание текстовых участков с помощью кнопок на этой панели может повредить выполнению программы. В нашем случае по умолчанию остается Enabled = True.

Font

Свойство Font влияет как на шрифт «новорожденных» ЭУ, вносимых на форму, так и на отображение текста при выполнении метода Print.

Для справки: метод Print печатает текст напрямую в окне - не в текстовом поле, не в заголовке, не в галочках и даже не в кнопках - а прямо на рабочей поверхности окна.

Вопрос: Где это можно использовать, почему и чем это лучше?

Ответ: Каждый ЭУ, помещенный на форму, отнимает у ОС определенное количество ресурсов, которые освободятся лишь при выгрузке формы из памяти. Такая выгрузка случается при:

- Unload AnyFormName
- End

В первом случае экземпляр окна «умирает навсегда». Если окно содержало результаты длительных вычислений, лучше его прятать при помощи оператора Hide: AnyFormName.Hide, а затем снова показывать при помощи Show.

Во втором случае программа прекращает работу.

Метод же Print таковых не использует, поэтому его можно использовать для вывода служебной информации, например, при выводе текущих результатов процессов - копировании файлов с отображением их имен на специально приготовленном окне.

По правде сказать, я нигде не использую Print.

Существует нюанс в выборе шрифта. Я редко использую системные шрифты, предлагаемые по умолчанию Бейсиком, по одной простой причине.

Однако предоставлю Вам возможность самолично убедиться в неосмотрительности разработчиков ОС: попробуйте задать шрифт MS Sans Serif размером в 7, 11 пунктов для cboFonts или любого другого ЭУ.

Совет: *Arial есть практически у всех, кто под Виндой, а тот, кто не под Виндой, вряд ли нуждается в Ваших разработках. К тому же последние пяток лет Arial - юникодовый товарищ, так что Вы, фактически избавляетесь от проблемы несовместимости шрифтовых наборов. Если же Вы вдруг столкнулись с такой проблемой, как "крокозябры" - используйте Ресуры и строчные таблицы (о них и о способах их использования читайте в следующих номерах). Указав для каждого языка свою ресурсную строку, Вы практически локализуете софт под каждого, кто посмел им воспользоваться. Это лучше, чем советовать пользователю пересмотреть свои таблицы подстановки шрифтов, указанные в системных ини-файлах. Ведь большинство юзеров недооценивают «продвинутые» методы юзания ОС.*

ShowInTaskBar

Посмотрите вниз своего экрана. Представьте, что среди текущих задач Вы видите кнопку для панели инструментов. Странно, да? Тогда установите значение в False.

StartupPosition

Это свойство отвечает за первоначальное месторасположение окна при его первой загрузке. Выберите из списка то, что Вам больше по душе - можно задать расположение «визуально» (manual) при помощи Form Layout Window, но тогда, если Вы не учли, что еще существуют мониторы намного меньших размеров, нежели Ваш, Вы рискуете сделать форму недоступной для пользователя. Не повторяйте чужих ошибок - если форма велика, то все параметры (высота, ширина, X, Y) лучше производить математически, а не на глаз.

Visible

Можно, конечно, прятать форму, меняя ее свойство Видимость. А зачем? Есть определенные операторы Unload, Hide… Проверьте, однако, чтобы Visible осталось со значением True.

WindowState

Здесь имеется в виду, развернуто окно, свернуто оно или находится в нормальном состоянии. Не вижу большого смысла в тривиальном сворачивании окна Панели Инструментов, а также ее разворачивании. Однако наверняка есть любители поизмываться над юзером…

Я же оставляю Normal.

Кнопки

Итак, плавающая панель теперь действительно плавает, будучи немодальной, и вызванной с указанием ее «родителя».

Примечание: MyCompad работает с файлами по принципу Блокнота - можно работать только с одним файлом в одном процессе (Instance - см. следующие публикации). Для того, чтобы загрузить, к примеру, второй файл, нужно создать еще один экземпляр программы. Такой интерфейс называется Single Document Interface (SDI). Существует также другая концепция работы с документами - а-ля MS Word. Такой подход позволяет одновременно работать с несколькими файлами и называется, соответственно, Multiple Document Interface (MDI). В MDI существуют такие понятия как MDIChild («Ребенок») и MDIForm - иными словами, одна форма порождает другую. Наше упомянутое понятие "родитель" не имеет ничего общего между MDI-субординативным рангом.

MDI-проект будет рассмотрен подробно в дальнейших уроках.

Теперь можно наполнить форму-контейнер контролами. Но сперва придется поразмыслить, какие команды можно вынести как наиболее часто используемые в процессе работы с текстом.

Очевидно, что наиболее важными и нужными командами окажутся следующие:

- Копирование
- Вставка
- Просмотр буфера
- Удаление выделенного участка
- Поиск
- Замена Выбор начертания и размера шрифта для удобства просмотра

В прошлых уроках была рассмотрена реализация Буфера Обмена (Clipboard), а именно определение типа данных, содержащихся в нем, вставка и копирование. Весь соответствующий код выполняется при нажатии на меню. А раз уж мы используем меню в качестве главного хранителя кода, то, разместив на форме кнопки, нам останется только связать их нажатия с нажатиями на соответствующих пунктах меню.

Для удобочитаемости кода назовем кнопки по аналогии с именами меню:

- cmdCopy
- cmdPaste
- cmdShowClipboard (Мой проект MyCompad`а не имеет меню mnuShowClipboard - я довольствуюсь быстрым вызовом через кнопку на панельке, однако Вам ничто не мешает создать его самостоятельно).
- cmdDelete
- cmdFindReplace

Теперь добавим ComboBox, переименуем его в cboFonts, и TextBox - назовем его txtSize.

Дело сделано, форма принимает классический вид панели инструментов, однако перед наведением последнего предпраздничного марафета нужно кое-где поковыряться.

В таблице приведены важнейшие из свойств cboFonts и txtSize.

CboFonts

Свойство	Значение
Sorted	True
Style	2 - DropDownList
ToolTipText	Нет

TxtSize

Свойство	Значение
HideSelection	False
MaxLenght	2
Multiline	False
Text	Нет

Описание свойств cboFonts

Свойство **Sorted** (*Сортировка*) упорядочит по алфавиту шрифты в списке, Style (Стиль) - задаст нужный стиль списку, не позволяющий вводить с клавиатуры имена несуществующих шрифтов (на данном этапе нам это просто не нужно. Далее будет рассмотрена функция «автокомплита» списка по мере нахождения похожих символов), TooltipText - это подсказка, появляющаяся при наведении курсора мыши на ЭУ. В MyComPad`е *ToolTipText* будет использован для отображения слишком длинных имен шрифтов, которые могут оказаться нечитаемыми в маленьком пространстве списка. А поскольку на стадии разработки мы не в силах добавить ни одного элемента в список шрифтов (можно, конечно, но… так не настоящие программисты не поступают), то список и подсказки будут определяться динамически.

Описание свойств txtSize

HideSelection (*Прятать Выделение*). В принципе, не критично, если Вы оставите значение по умолчанию, а именно False. Тогда Вы (и пользователь MyComPad`а) не будете видеть, какой элемент управления в данный момент находится в фокусе (является активным).

MaxLenght (*Максимальная длина текста*). Я поставил **2**. Это значит, что максимальный кегль (размер шрифта в пунктах) составит две максимальные цифры. Угадайте-ка, какой максимальный размер шрифта будет в программе!

Multiline (*Многострочность*). Идея этого маленького окошка состоит в том, чтобы дать пользователю возможность вводить значения размера шрифта с клавиатуры, подтверждая символом с ascii-кодом 13 (Enter). Конечно, можно разрешить и абзацы…

Итак, заполним-ка список экранными шрифтами!

Разумеется, такие процедуры следует делать при первом же запуске окна. Поэтому подготовим себе рабочую заготовочку для кода, выбрав из списка доступных событий и методов **Form_Load**:

```
Private Sub Form_Load()
   Dim fntCnt As Integer
   For fntCnt = 0 To Screen.FontCount
       cboFonts.AddItem Screen.Fonts(fntCnt)
   Next

   cboFonts.RemoveItem 0
   If FontExists(txtMain.Font) Then Call SetFontList(txtMain.Font)
End Sub
```

Операционная система - это набор библиотек, системных файлов и Форточного балласта, благодаря которому программы взаимодействуют между собой, периодически обмениваясь данными (сообщениями) и вылетая. Благодаря появлению такого понятия как Объектно-Ориентированное Программирование стало возможным упрощение почти всех рутинных задач на любой ОС. Трудно себе представить, на каком уровне осталась бы компьютерная индустрия, если бы не новый подход к программированию. Однако до темы «ООП и ООД» еще весьма далеко. Поэтому скажу лишь вот что: Существует понятие Объект. Объектами являются принтеры, экран, Ваша программа MyComPad. Первые два располагают наборами шрифтов, которые можно перечислить по целочисленным индексам. А для того, чтобы приказать приложению пройти весь путь от ноля (в массивах 0 является первым элементом) до максимального (последнего) значения такого индекса, нужна особая система, - лаконичная и логически завершенная.

Одной из таких структур, называемых циклами, является For…To… …Next. Дословно это переводится как *[От…До… …Хватит!]*

Итак, два объекта. Наша цель - указать программе шрифт для отображения его на экране, посему логично было бы выбрать из двух объектов именно **Screen**. Так получилось, что в отличие от большинства коллекций (наборов) - опять-таки, ООП!!! - количество шрифтов и идентифицирующее, функционально индексированное свойство, которое и нужно использовать, - расположены за милю друг от друга! Поэтому для получения количества начертаний используем FontCount, а для перебора - *Fonts(переменная-индекс)*.

По завершении цикла, когда достигнут **FontCount**, удаляем первый элемент списка, - пустую строку в комбобоксе, от которой мы не в состоянии избавиться на стадии разработки приложения. Дальнейшие действия - согласовать имя шрифта с тем шрифтом, который используется в текстовом поле txtMain. Здесь, пардон, два выхода: либо наобум (мы-то помним - Arial!) либо спросить у самого txtMain, что он использует, и затем, если в списке такое обнаружено, - установить текст в комбобоксе, причем напрямую его текст не изменить - read only property… Значит, обойдемся лишь индексом элемента списка. Как видно, использованы две функции - одна функция-верификатор, возвращающая значение Boolean, другая - обычная процедура.

Читатель спрашивает, каким образом можно соорудить ***свои, чисто пользовательские кнопки****.*
Раз уж у нас с Вами вошло в привычку сперва разъяснять принципы, а уж затем - детали, сегодня мы поступим так же. По существу, кнопка - не важно, как она выглядит, - это контейнер с двумя картинками, каждая из которых олицетворяет одно из двух возможных положений: нажатое и отжатое. Исходя из этого, можно манипулировать этими картинками как угодно, лишь бы все это выглядело по-человечески и пользователю не хотелось убрать софтину с глаз долой.

Приготовление изображений

По правде говоря, не вижу лучшего способа "выцепить" где-то кнопку, чем клавишей PrintScreen (с альтом - только активное окно) заполучить саму картинку с кнопкой, в Photoshop`е обрезать (выделить что нужно и - Alt+IP или Image>Crop), затем, в случае необходимости, вытереть внутренности самой кнопки, оставив лишь ободок, сдублировать слой, развернув его на 180 градусов - вот Вам и нажатое состояние! Затем добавить еще один слой и уже на нем рисовать чего Вам заблагорассудится. В дальнейшем, используя формат PSD (для сохранения слоев, естественно), отключая и включая слой с "нажатой" картинкой, можно с легкостью создавать все новые и новые кнопки, сохраняя при этом те же пропорции и стиль, что только прибавит Вашему ПО положительных черт в глазах пользователя.

Реализация кода. Вариант 1 - попроще

Итак, помещаем на форму три (! Три !) элемента управления Image. Это не единственный элемент управления для оперирования изображениями, однако он полезен ровно в той степени, в которой нам необходимо для наших целей. ЭУ PictureBox обладает рядом полезных свойств и методов, которые более пригодны для рисования, чем для хранения информации о картинке. Не следует забывать, что три изображения для одной кнопки - не так уж и много, однако если у Вас не одна кнопка и не пять, а 20-25 - то размеры скомпилированного исполняемого файла скажут сами о себе, и, кроме всего прочего, пострадает быстродействие... Поэтому выбираем три ЭУ Image. Впрочем, решать Вам.

Один элемент управления следует разместить там, где будет находиться кнопка. Остальные два - не важно где, потому как свойство Visible мы все равно устанавливаем в False. Я же всегда растягиваю форму по высоте и ширине так, чтобы были видны все спрятанные элементы, но указываю параметры окна в процедуре Form_Load (загрузка формы). Однако не стоит забывать о том, что некая API-функция ShellExecute требует указания параметра состояния окна вызываемой программы, например, Maximized, Minimized, Normal. Вывод таков: не указав свойство Visible = False, вы рискуете показать пользователю всю кладовку, если тот запустит Вашу программу с аргументом, например, Maximized...

Итак, назовем первую картинку cmdOK. Остальные две - cmdOKup и cmdOKdown. Надеюсь, имена компонентов не введут Вас в тихий шок завтра-послезавтра, когда Вы немного "отойдете" от процессаJ.

Помещаем в первую изображение с отжатой кнопкой, а во вторую - с нажатой. Одним словом, у нас есть два кармана с "раздеребаненной" зарплатой. Один - для себя (на новую материнку, СиДи-Писалку и новый Visual Studio) , другой - для жены и деточек (на мелкие расходы - конфеты и мороженое)... В разных ситуациях мы тянемся в соответствующий карман. В общем, ничего сверхъестественного: для изображений (а именно для ЭУ Image) предусмотрены события MouseDown и MouseUp. Существует еще и Click с DblClick, однако в параметрах этих событий (процедур событий, если быть точным) отсутствуют такие аргументы как Button, Shift, X и Y. Это говорит о том, что софтинке будет абсолютно "по цимбалам", какой кнопкой мыши Вы клацнули по "кнопке", какую кнопочку на "клаве" удерживали, и уж тем более - географическое положение... Нет, не Ваше, а указателя мыши. В отличие от Click и DblClick, у нас в распоряжении переменные Button, Shift, X и Y. Признаюсь, не довелось мне на практике использовать X и Y - ну не нужны мне координаты при кликах и все тут!

Так вот, дважды-кликаем на основной кнопке cmdOK. По умолчанию открывается процедура-обработчик cmdOK_Click. Из списка доступных для Image событий выбираем MouseDown. Полученный шаблон модифицируем таким образом:

```
Private Sub cmdOK_MouseDown(Button As Integer, _
          Shift As Integer, X As Single, Y As Single)
      cmdOK.Picture = cmdOKdown.Picture
  End Sub
```

```
' а для отжатия кнопки - так:
Private Sub cmdOK_MouseUp(Button As Integer, _
        Shift As Integer, X As Single, Y As Single)
    cmdOK.Picture = cmdOKup.Picture
End Sub
```

Теперь целесообразно скопировать тело обработчика события MouseUp в процедуру Form_Load - при запуске кнопка должна выглядеть по крайней мере адекватно, если только Ваш образ мысли соответствует принципам планеты Земля.

Если Вы не желаете никакой реакции лже-кнопочек на правые щелчки, добавьте **If Button = 1** Then перед каждой строкой присвоения изображений, другими словами, задайте им всем проверочку... Если Вы хотите, чтобы программа различала кликанье с Шифтом и без него, приведите код к такому виду:

```
Private Sub cmdOK_MouseDown(Button As Integer, _
     Shift As Integer, X As Single, Y As Single)
  Select Case Shift
  Case 0
     If Button = 1 Then _
     cmdOK.Picture = cmdOKdown.Picture
  Case 1
     MsgBox "Shift!"
  Case Else
     Exit Sub
  End Select
  End Sub
```

Значения аргумента Shift включают в себя не только одноименную клавишу - здесь собраны и "альты", и "шифты", и "контролы", и их всевозможные комбинации. Я не стану Вам перечислять их - Вы все равно их не запомните, да и ни к чему это. Просто добавьте еще одну временную кнопку (теперь можно не извращаться - добавьте стандартную), подготовьте для нее код:

```
Private Sub Command1_MouseDown(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
      MsgBox Shift
End Sub
```

Пощелкайте кнопочку с различными комбинациями нажатых функциональных клавиш и все поймете сами. Это иногда лучше, чем нравоучения при дефиците площади, не так ли?

Реализация кода. Вариант 2 - это уже что-то

Представьте себе ситуацию, когда пользователь выбирает элемент в списке, содержащем гиперссылки, готовящиеся к загрузке из Интернет, затем нажимает "Грузить", а потом, когда программуля начала свое действо, нажимает, например, "Редактировать" или "Удалить".

В общем, смысл понятен? Да, нужно реализовать обыкновенную блокировку доступа к некоторым кнопкам. А раз мы не настолько ленивы, чтобы ограничиться определением свойства Enabled, то почему бы не добавить к двум сделанным нами картинкам для кнопки еще и третью - для состояния "Disabled". Конечно, такого состояния вы не найдете для ЭУ Image - применять третье изображение придется наряду с изменением свойства Enabled. Зато пользователь, увидев даже визуально "отрубленное" устройство, даже не попытается по нему клацнуть. Рассчитывать ведь нужно на разный контингент...

Теперь прошу представить еще одну ситуацию, когда у Вас как

минимум столько кнопок, сколько у Вашего Ворда на "фейсе" - около пятнадцати при средней паршивости диагонали монитора. Представьте себе, сколько необходимо написать кода для каждой кнопки! Даже используя копирование/вставку с заменой имен в сиснтаксических джунглях, задача будет не из легких. Поэтому открою Вам одну из великих тайн, познав которые, "салабоны" превращаются в гуру программирования, и на которых держится программирование вообще. Так, программирование тысячи и одной кнопки можно выполнить фактически в четыре строки кода! Не верите? Тогда читайте далее!

Массивы

Мы все ежедневно встречаемся с аналогами массивов, сами того не замечая. К примеру, покупая билет в кино (ну да, когда это было...) - имеем дело с двумя аргументами: рядом и местом в зале, или отыскивая по карте те или иные участки Земного Шара - широтой и долготой, наконец, отправляя почту (традиционную, неэлектронную), мы указываем государство, город, улицу (или проспект), дом, квартиру. Вы наверняка заметили разницу в количестве аргументов: в первых примерах хватило двух параметров, чтобы описать цель, в последнем - аж пять. В таких случаях говорят о массивах двумерных и многомерных. Так, если бы страны были кем-то пронумерованы числами типа Integer или Long, причем Украина - под номером 3, а расположенные в них города - аналогично, (и так далее), то свой адрес я мог бы в виде массива определить как MyAddress = Human(3, 0, 50, 4, 4). Если Вас пугает нолик в приведенном примере - привыкайте к мысли, что все в массивах (и коллекциях/наборах, но об этом - в теме "ООП") нумеруется с ноля. И если Visual Basic до шестой версии позволял определение начала нумерации массивов, как никакой другой язык, то VB .NET этот допуск уничтожил, - и поделом: хватит путаницы! Это не столь существенная помощь в программировании, чтобы ставить под удар надежность создаваемого ПО.

Стало быть, новое тысячелетие - с правильным видением массивов. Ура. Я же не рекомендую использовать нумерацию с единицы, даже если у Вас версия Бейсика еще с того тысячелетия (6-я), или Word97, и только поэтому не скажу, как эти правила меняются...

Таким образом, 3 - государство, 0 - город, 50 - улица (J), последние два аргумента - дом и квартира. Ну, двух-, трех- и "более-мерные" массивы применимы скорее к построению сложных конструкций, (пример: 3D-графика, ее движок), а не к кнопкам, поэтому уделим внимание обычным одномерным массивам, где элементы определяются единственным показателем индекса этого массива:

```
MyHyperlink(219).DownLoad
   или так:
Dim i As Integer
   For i = 0 to Val(Text1)
       List1.List(i) = "http://" & List1.List(i)
       List1.ListIndex = i
   Next i
```

В этом примере каждому элементу списка (читай: массива) слева вставляется текст.

Ресурсы

Итак, наша цель - динамически присвоить кнопкам. Для этого нам необходимо наловчиться пользоваться файлами ресурсов (.res), которыми орудуют программисты на Си и Делфи как дважды два. Раньше программисты на Бейсике запускали специальные программы для записи/чтения из файлов Ресурсов. Сейчас же VB6 распространяется со встроенным Add-In`ом VB Resource Editor. Чтобы его вытянуть на панель IDE Бейсика, щелкните меню **Add-Ins>Add-In Manager...**, затем выберите в списке указанную строку и кликните OK. Далее - щелкаем на появившейся зеленой пиктограмме редактора ресурсов, после чего нам откроется главное окно Редактора Ресурсов. Если честно - элементарная штука, упростившая многие задачи не только мне - всему миру программистов на любом языке программирования. Кроме того, некоторые визуальные оболочки у Вас и не спросят, как Вы предпочитаете хранить изображения - непосредственно где-то под рукой (в VB можно на форме), либо в res-файле. Между прочим, файлы frx, которые часто сопровождают файлы форм (frm) - это данные и для картинок, и для другой двоичной используемой в софтине дряни, если только Вы ее не запихнули в Ресурс.

В Ресурс можно включать четыре основных вида информации плюс один пользовательский (т.е. неопределенный, любой):

строки (на очень многих языках, причем в зависимости от ОС и ее настроек эти строки будут взаимозаменяться в приложении. Например, метка с надписью "Опасная зона" могла бы выглядеть следующим образом: "Danger zone", причем автоматически - от Вас ничего не требуется! Посмотрите, как работают стандартные диалоговые окна, надписи в стандартных Windows-приложениях, однако не забывайте о синдроме Word97 и никогда не ставьте комбинации клавиш в зависимость от Caption пунктов меню ...);

- изображения (пригодны к использованию лишь .bmp-файлы);
- пиктограммы ("иконки", .ico);
- анимированные пиктограммы, или курсоры (.cur).

К пользовательским (Custom) относятся все другие файлы, которые можно поместить в .res.

Поскольку цель сегодняшнего заседания - научиться создавать кнопки и быстро их загружать, то ограничимся лишь типами Picture и String - для подсказок (Tooltips).

Открою секрет: идентификаторы ресурсов в Resource Editor`е могут и не следовать в строгой очередности, т.е. после первого (или второго) может идти две тысячи третий. Это и толкнуло меня к мысли, что если выделить для пятнадцати кнопок первые 15 пунктов каждой из трех первых сотен с учетом изображений для "Disabled", то серьезно экономится время выполнения за счет хорошей реализации работы с Ресурсами (API, друзья мои, практически всегда таковым и является), плюс получаем лаконичнейший код (есть ли такое слово?):

```
Public Sub setButtons(Count As Integer)
  With Form1
  Dim i As Integer
       For i = 1 To 15
             Load .cmdImage(i)
            .cmdImage(i).Left = .cmdImage _
            (i - 1).Left + .cmdImage(i).Width
            .cmdImage(i).Picture = _
            LoadResPicture(100 + i, _
            vbResBitmap)
            .cmdImage(i).Visible = True
            .cmdImage(i).ToolTipText = _
            LoadResString(i)
       Next i
  End With
  End Sub
```

Объясню:

```
Для Формы1
От 1 до 10
Создать_Копию_Кнопки (С_Текущим_Счетчиком_Индексом)
Положение_Слева = Положение_Слева_Предыдущей_Кнопки + Ширина
Картинка = Загрузить_Из_Ресурсов(Картинка_Для_Отжатой, По_Формату)
      Сделать_Ее_Видимой
      Текст_Подсказки_Тем_Же_Образом
    Конец_Цикла
Сформой1Покончено
```

Если учесть, что со сто первого по сто пятнадцатый пункт Ресурсного пространства - это "отжатые кнопки", а с 201 по 215 - "нажатые", и существуют строки с 1-й по 15-ю, то все выйдет прекрасно, и Ваши кнопки больше от Вас не зависят - процедура содержит аргумент Count,

через который Ваше общение и ограничится: Call SetButtons(15). Ее можно модифицировать, указав как аргументы и имена формы, кнопок, отступы как разделители, - тогда это будет переворот в мире рац-программирования. (Термин я придумалJ). Если же необходимо изобразить отступы (Separators по-крутому), например, после 4 и 9, то пишем так:

```
If i = 5 Or i = 10 Then
      .cmdImage(i) = …..<дальше - так же> + 80
  Else
      <а здесь - в точности, как указано выше>
  End If
```

Фокус в том, что 5 и 10 встретятся всего один раз, так что смело применяйте мою тактику. 80 можно заменить на иное число - по вкусу, но можно разделить оставшееся свободное место на количество промежутков. Если Ваши кнопки должны выстроиться еще и по вертикали - мне Вас жаль. Но, повозившись, наверняка найдете свои ошибки. В любом случае используйте свойства Height и Top и абсолютно всегда отталкивайтесь от ближайшего соседа.

Я поместил эту функцию в модуль (.bas) и использую просто выполняя обращения:

[ГрафическийЭлементУправления.ЕгоКартинка = LoadResPicture(Индекс, vbResBitmap)] - для изображений, и

[ТекстовыйУправления.ЕгоТекст = LoadResString(Индекс)] - для текстовых полей, текстовых переменных и т.д.

Таким вот хитрым образом я загружаю "нормальные состояния" кнопок. При нажатии левой кнопкой мыши (используем рассмотренную процедуру) VB автоматически вписывает аргумент Index для массива элементов управления (ведь должны мы их как-то различать?):

```
Private Sub cmdImage_MouseDown(Index As Integer, Button As Integer, _
                                             Shift As Integer, _
                                             X As Single, Y As Single)
  If Button = 1 And Shift = 0 Then
   cmdImage(Index).Picture = LoadResPicture (200 + Index, vbResBitmap)
  End If
  End Sub

  Private Sub cmdImage_MouseUp(Index As Integer, _
               Button As Integer, Shift As Integer, X As Single, Y As Single)
   cmdImage(Index).Picture = LoadResPicture(100 + Index, vbResBitmap)
End Sub
```

В процедуре MouseUp я сознательно "забыл" проверить кнопку мыши и Шифты - таким образом пользователь не оставит кнопку нажатой, динамически сменив щелчок с Click`a на RightClick.

Для недоступных кнопок код аналогичен, только вставлять его надо непосредственно в той процедуре, которая перекрывает доступ к кнопкам и другим элементам управления, и вместо 200 или 100 приемлемо 300.

Вот и все. Осталось только научить Вас создавать массивы. О, да это - самое простое в данной истории. Если создается массив элементов управления - просто скопируйте его и вставьте в ту же форму, и ответьте положительно. Это - простой путь. Ну, а сложный (пожалуйста, барабанную трель...) заключается в установке значения Index любого элемента управления в ноль. Согласитесь, и тот, и другой способы весьма тяжелы для простого обывателя, но, я уверен, у Вас получится. Да, чуть не забыл. Нулевой элемент все-таки базовый, и использовать его просто неудобно. Рекомендую сделать его невидимым.

Если же это - массив символов, то его создание происходит так:

Dim MyString As String(15, "w"), после чего имеем 15 буквочек "даби". Затем можно с той же легкостью доступиться до каждой из них через индекс:

MyString(2) = "a". Здесь мы заменили второй элемент на "а".

Все гениальное - просто...

Мораль басни

Ведь не зря придумали массивы. Это и просто (в случае с одно- и даже двумерными), и приятно (ну, кому как...), и ... рационально. Приелось слово?

* * *

Вот и вся история. Не для смеха была рассказана, а для урока. 3000 компонентов... Да кто знает, сколько будет в Вашей программе? Другими словами, рекомендую к использованию именно массивы, где только можно. Но только до тех пор, пока мы не коснулись ООП. Хотя, если с умом...

Текстовый редактор

Повседневные задачи офисного сотрудника, верстальщика, дизайнера (в большей части с уклоном на веб-аспект), да и просто домашнего пользователя зачастую сводятся к оперированию данными, главным образом текстовыми. Посему продолжение многими ожидаемый цикл "Мышление в стиле Visual Basic" я ознаменую продолжением эпопеи MyComPad`а - текстового редактора, корторый мы собирались (и до сих пор это делаем) сделать редактором HTML-страниц по умолчанию (для пользователей IE5), который, как и было анонсировано, будет превосходить стандартный НотоПад из арсенала Microsoft Windows.

Для вразумительной беседы нам с Вами потребуется поднапрячься, и, на фоне вступающей в силу Весны (статья готовилась в мае '2001), все же припомнить некоторые ключевые моменты в создании Вин-приложения на Visual Basic. Естественно, в ходе изложения этого увлекательного метериала мы будем то и дело вспоминать уже пройденное - буквально поверхностно. Каждый отдельно взятый "урок" (именно "уроками" называются статьи из цикла на сайте vbag.hypermart.net) отныне будет самостоятельным, однако иногда опирающимся на уже освоенные темы. Сложного здесь, в принципе, ничего не будет, за исключением API-вызовов, которые знать на зубок не нужно - можно лишь иметь под рукой сводку этих самых Системных Функций. Текстовый файл Win32API.txt, который используется для "выдирания" API-объявлений, констант и типов, входит в любой комплект Visual Basic. Однако необходимо остерегаться фривольного применения такого материала, так как файл содержит некоторые ошибки, многие из которых не являются (и не могут) "очень фатальными" для Системы, однако неработающий код - это не есть хорошо. Поэтому и на сайте, и на страницах "Моего Компьютера" вы найдете только проверенный код.

Начиная с данного урока я подразумеваю, что читатель уже знаком с понятием API. Те, кто по каким-либо причинам не застал предыдущих публикаций на эту тему, могут найти ее на vbag.hypermart.net - тема API весьма обширна, и повторение столь большого объема информации на страницах "Мего Компьютера" мне не представляется разумным. К сожалению, газета не предусматривает бесконечные листинги на своих страницах, поэтому ищите теорию в газете, а полнофункциональные архивы проектов VB 6.0 - на моем сайте. Кроме того, я никогда не игнорирую письма доброжелательного читателя. Частенько я создаю проект любознательному отправителю на рассмотрение. Недоброжелателям же лучше воздерживаться - я не люблю критику :) Шутка. Все замечания, пожелания, вопросы и проблемы изливайте на **ag@ukr.net**.

Итак, уясним, что должен уметь MyComPad:

- Открытие/Сохранение текстовых и гипертекстовых файлов,
- Поиск/Замена,
- Форматирование для отображаемого текста (Шрифт, размер),
- Дополнительные "прибамбасы" типа элементарной статистики;
- Предварительный просмотр HTML в броузере,
- Контекстные меню в Проводнике (т.е. ассоциация соответствующих форматов с MyComPad`ом, Default HTML Editor);
- Драг-Н-Дроп файлов в ТекстБокс;
- "Продвинутые прибамбасы" типа шаблонов HTML-разметки, плагины (свободного стиля, господа, это Вам не Photoshop).

Открыть/сохранить: что есть проще?

Наверняка кто-нибудь помнит, что мы поместили на форму frmMain элемент управления MS Common Dialogs Control и назвали его CD (здесь я вынужден повториться: избегайте длинных имен компонентов - это неудобно, и ни о чем не говорящих - через некоторое время Вы напрочь забудете, что это за фрукт. Читайте о системе именований на моем сайте).

Открытие текстового файла - одна из задач, превалирущих над другими при создании всякого рода редакторов, утилит, сохраняющих свои установки, и тому подобных.

Как известно, ничего не бывает "просто так". Тем более в такой непростой игрушке, как Win9x. В ней все подчистую именуется своими уникальными идентификаторами, номерами и т.д., и файлы - не

исключение. Все дело в "закупоренности" Visual Basic. Программисты на Си/Си++ знают, что такое потоки. Мы же, "прикладные" программисты на "Басике", и знать не хотим, что это такое. Однако...

Перед тем, как открыть любой файл (режимов открытия файлов всего три. В нашем случае - "текстовом" - все как никогда просто) мы обязаны

- освободить переменную типа Целое (Integer) чтобы Visual Basic мог автоматически идентифицировать (просится на язык слово "поток", однако точнее и не выразиться) файл, открытый этой программой. Поэтому никакого труда не составит, например, не закрывая файлы, управлять ими: в этот напечатать (в смысле добавить строку или заменить содержимое - об этом позже), из этого - наоборот, прочесть столько-то символов начиная с такой-то позиции. Что такое "освободить"? Да все просто: создать и указать, что она - следующее свободное число;

- Создать две переменные типа Строка (Srting): одну - для глобального хранения строк, другую - как челнок, "черпачок" для вычерпывания всего объема. Таким образом, более "мелкая" переменная то наполняется из построчного считывания из файла, то выплескивает свое содержимое в общий котел, а затем опустошается. Кроме всего прочего, метод построчного считывания является, на мой взгляд, наиболее простым способом контроля за содержимым текстового объема - это действительно проще, чем создавать функции-парсеры!;

- Дойдя до конца файла (условие **EOF**, *End Of File*), присваиваем свойству Text текстового элемента управления, например, txtMain, значение [Переменная] .

```
Dim i As Integer
Dim strPortion As String
Dim strAll As String
Dim Result
i = FreeFile
Open "c:\Autoexec.bat" For Input As #i
While Not EOF(i)
     Line Input #i, strPortion
       Result = MsgBox("Эй, нам нужна эта строка?" _
       & vbCrLf & vbCrLf & _
       strPortion, vbYesNoCancel + vbInformation, _
       "ну так что будем делать?")
     Select Case Result
     Case vbYes
          strAll = strAll & vbCrLf & strPortion
     Case vbNo
          ' бъем баклуши
     Case Else
          ' В нашем случае Иначе - Отмена
     End Select
Wend
Close #i
```

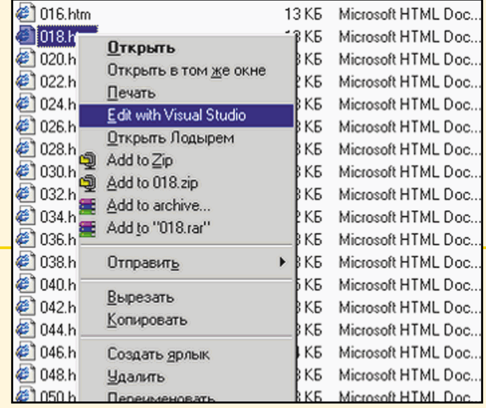
К слову, данная техника оперирования текстовыми данными - обыденность для программистов любого уровня и была тщательна мною рассмотрена ранее. Намного интереснее обстоят дела с тем Как-Узнать имя файла, который нам нужно открыть. Тут есть два варианта: либо мы подряжаем для этих целей Windows Explorer, либо просим пользователя указать файл явно - то ли в текстбоксе, то ли узнав от CD (*См. предыдущие уроки о Common Dialogs Control*) то ли вообще при помощи InputBox:

Далее - как в предыдущем примере. Однако не забудьте прописать все указанные в предыдущем примере переменные. А еще лучше - нажмите Ctrl+Home, Enter и введите **Option Explicit**. Затем найдите время, чтобы

в настройках IDE Visual Basic внести директиву "Обязательное объявление переменных" - чтобы каждый раз не вводить директиву с клавиатуры. Это сэкономит Вам время и не испортит отношения к VB 6.0.

```
Dim strFileName  As String
strFileName = InputBox("Имя!", "имя файла", "c:\Autoexec.bat")
i = FreeFile
...
```

Ассоциации, ассоциации...



Для того, чтобы Проводник имел возможность передавать нам имена файлов, можно либо вручную (используя Regedit.exe), либо "вообще вручную" - :) - используя клавиатуру и созданный этим орудием файл .reg, либо при помощи специальной API (что тоже неплохо) ассоциировать любой файл с MyComPad.

В случае с reg-файлом все просто:

```
REGEDIT4
[HKEY_CLASSES_ROOT\txtfile\shell\open\com-
mand]
@="C:\\WIN98SE\\NOTEPAD.EXE %1"
```

Впрочем, лучше исследуйте Реестр на предмет ассоциаций, и, сделав выводы, поймете, что, в сущности, функциональными в этом случае есть только два ключа. Для PlainText-файлов это ключ .txt и txtfile, который указан в параметре по умолчанию первого. Таким образом, перенаправив ссылку от txtfile к своему собственному ключу, вы избавите текстовые файлы от контекстных меню Visual Studio, например. Добавив же меню, Вы расширите выбор пользователя. Значение по умолчанию в ключе txtfile будет срабатывать по двойному щелчку манипулятором типа "мышь" в менеджере файлов, или Проводнике - у кого как :). Если вместо полного пути к исполняемому файлу здесь прописать пароль доступа в Интернет, телефон доступа, прокси, порт и логин, то тройные щелчки будут конвертировать txt в rtf, а четверные - в html с JavaScript и немного Flash5. Не верится? Правильно. Четырех щелчков будет недостаточно :).

Можно обеспечить периодическое ассоциирование файлов с программой - в этом есть некий смысл: Вы перемещаете EXE, пути в Реестре уже не соответствуют реальному пути, однако после первого запуска все становится на свои места. Для этого достаточно в процедуру Form_Load() поместить вызов подготовленной функции:

```
Dim FAResult As Boolean
FAResult = AssociateFile("txt", App.Path & "\" & _
   LCase(App.EXEName) & ".exe", "Файлики MyComPad`a")
```

Как видно из примера, регистрировать можно что угодно. Можно ассоциировать, например, диски (ключ Drive в HKEY_CLASSES_ROOT) со звуко- или MIDI-редактором. Тогда из HD можно получить CDA-драйв :). Шутка.

Тело самой функции **AssociateFile** выглядит совсем не так привлекательно, как ее вызов. Однако, как говорится, любишь кататься, люби и самочек возить... Здесь используются все приколы для юзания Windows Registry по-взрослому. Поэтому рекомендую саму функцию вынести куда-нибудь в модуль .bas, и использовать где Вам только нужно будет ассоциирование файлов с форматами (точнее, расширениями) в любых последующих проектах.

Таким вот дивным методом мы выполняем весь этот кошмар (однако всмотритесь в приведенный код, и через 15-20 минут он уже Вам не будет казаться столь ужасным. Более того, через некоторое время Вы заметите кое-какие закономерности и сможете

```
Declare Function RegCreateKey Lib "advapi32.dll" _
   Alias "RegCreateKeyA" (ByVal hKey As Long, _
   ByVal lpSubKey As String, phkResult As Long) As Long
Declare Function RegSetValue Lib "advapi32.dll" _
   Alias "RegSetValueA" (ByVal hKey As Long, _
   ByVal lpSubKey As String, ByVal dwType As _
   Long, ByVal lpData As String, _
   ByVal cbData As Long) As Long

Const ERROR_SUCCESS = 0&
Const ERROR_BADDB = 1&
Const ERROR_BADKEY = 2&
Const ERROR_CANTOPEN = 3&
Const ERROR_CANTREAD = 4&
Const ERROR_CANTWRITE = 5&
Const ERROR_OUTOFMEMORY = 6&
Const ERROR_INVALID_PARAMETER = 7&
Const ERROR_ACCESS_DENIED = 8&
Const HKEY_CLASSES_ROOT = &H80000000
Const MAX_PATH = 260&
Const REG_SZ = 1
```

Тело функции

```
Public Function AssociateFile(Extension As String, PathToExecute As String, _
      ApplicationName As String) As Boolean
            ' Функция возвращает True, если все OK. Если факир был пьян - False
        AssociateFile = False
        Dim sKeyName As String     'Ключ
        Dim sKeyValue As String    'Значение ключа.
        Dim ret&                   'Состояние ошибки, на всякий случай конечно
        Dim lphKey&                'Идентификатор созданного ключа Реестра

    'Создаем ключ
        sKeyName = ApplicationName
        sKeyValue = ApplicationName
        ret& = RegCreateKey&(HKEY_CLASSES_ROOT, _
                 sKeyName, lphKey&)
        ret& = RegSetValue&(lphKey&, "", REG_SZ, sKeyValue, 0&)

    'Создаем ключ для .TXT
        sKeyName = "." & Extension
        sKeyValue = ApplicationName
        ret& = RegCreateKey&(HKEY_CLASSES_ROOT, sKeyName, lphKey&)
        ret& = RegSetValue&(lphKey&, "", REG_SZ, sKeyValue, 0&)

    'Командная строка для исп. файла
        sKeyName = ApplicationName
        sKeyValue = PathToExecute & " %1"
        ret& = RegCreateKey&(HKEY_CLASSES_ROOT,  sKeyName, lphKey&)
        ret& = RegSetValue&(lphKey&, "shell\open\command",  REG_SZ, sKeyValue, _
                       MAX_PATH)
       AssociateFile = True
End Sub
```

самостоятельно прописать в Реестре Описание формата файлов) каждый раз при запуске программы. А вызов-то совем ничего: [MyVar = Associate(Parameters...)]

Не лишним будет напоминание, что длинные строки кода в VB можно разделять пробелом и последующим за ним символом подчеркивания. Существуют, однако, некоторые исключения, но о них лучше меня Вам расскажет Visual Basic IDE.

Рассмотренные фрагменты являются только одной стороной медали. Теперь нужно каким-то образом сообщить MyComPad`у о том, что мы взвалили на него дополнительную работенку.

Для того, чтобы софтинушка поняла Проводника, существует такая "вещица" как **Command$**.

Проведите эксперимент:

- Внесите строку MsgBox Command$ сразу под Private Sub Form_Load()
- Скомпилируйте свою программу и вынесите ее или ее ярлык на Рабочий Стол
- "Перетащите" на ярлык/программу любой файл.

Итак, Вы убедились, что контакт с двумя лагерями - 1) Проводник и 2) MyComPad.exe налажен. Теперь самое время поиметь с этого хоть что-нибудь.

Первым делом выносим код открытия файла в функцию - куда-нибудь в стандартный модуль, при этом в самой функции указываем не "Autoexec.bat", а FileName. Сейчас все станет ясно. Имя функции может выглядеть примерно так:

```
Public Function OpenFile(FileName As String) As String.
```

По нажатию клавиши Enter IDE завершает начатое Вами фразой "End Function", между прочим.

Затем возвращаемся в процедуру открытия (загрузки) главной формы - *frmMain* - и дописываем (можно после вызова функции ассоциирования файлов):

```
If Command$ <> "" Then
   Dim F As String
   F = OpenFile(Command$)
   txtMain = F
End If
```

Все. Текстовое поле txtMain - главная арена сражений с символами между пользователем и компьютером - содержит текст из файла, полученного из файл-менеджера.

Если мы пропишем в *Form_Load()* еще и

```
FAResult = AssociateFile("htm", App.Path _
     & "\" & LCase(App.EXEName) _
     & ".exe", "Страничка из Интернета")
```

то этим привяжем еще одну прослойку общества обитателей жестких дисков к программуле. Так, имеет смысл ассоциировать такие же "инетовские" .asp, которые после инсталляции "Фотосхопа" перестают быть гипертекстом, но "становятся" сохраненными настройками оного. Можно вместо .txt взять *.* и тогда...

Drag`N`Play

Самое замечательное в современных приложениях для Windows`9x, NT, W2K, а также MacOS и Рулез/Линух - это, конечно, интерактивность. Честно говоря, не представляется общение человека за клавиатурой с машиной, когда выполняется, к примеру, графическая задача, мультимедийная презентация и т.д. не говоря уже о том, что отуствие продуманных интерфейсов, во-первых, не выдвинуло бы ПК в ранг первого помощника человека в вычислительных прикладных задачах, подвергло бы сомнению применение персонального компьютера в таких отраслях как.... Ой, чего это я?

Короче, дело в следующем: необходимо обеспечить Драггинг, (естественно, с последующим Дроппингом) в рабочую область. При этом должно выполняться закрытие открытого файла (а он действительно открыт - мы имеем дело с моделью SDI: попробуйте-ка закрыть файл в Блокноте, не закрывая самой программы!) и открытие нового.

Если память Вам изменяет (или Вы просто не читали предыдущих уроков в стиле VB), то мотайте на ус: для того, чтобы быть в курсе, изменен ли файл, достаточно объявить переменную типа Boolean в разделе глобальных объявлений (под Option Explicit, если Вы-таки внесли эту строку) и изменять ее значение на False каждый раз, когда изменяется содержимое txtMain - оптимально сориентироваться на "рафинированной" процедуре *txtMain_Change()*? незачем шпионить за пользователем, до нас уже все тщательно продумано, и устанавливать в True, когда изменения сохранены. Естественно, при открытии файла переменная должна имитировать "сохраненное состояние" файла. Другими словами, пременная *FileSaved* = **True**. Однако некоторое время назад, когда мы с Вами только-только начинали работать над MyComPad`ом, была объявлена переменная **Dirty**, по смыслу противоположная изложенному. Что ж, в таком случае представьте себе, что мы говорим не о "сохраненности", а об "измененности". В таком случае в процедуре загрузки формы пишем *Dirty* = **False**.

Именно так, а не иначе, не то нетрудно будет заблудиться в логических лабиринтах - когда и что... Заметьте: Notepad.exe неизмененный ПУСТОЙ, т.е. только что созданный файл даже не предлагает к сохранению. Мало того, по ывполнении некоторых действий, если состояние "Безымянного" файла пришло к первоначальному облику, картина та же.

Для Драг-Н-Дроппинга файлов, директорий и дисков из внешних источников типа Explorer нам потребуется буквально следующее:

- Установка свойства главной рабочей области - текстового поля txtMain OleDropMode = Manual
- Создание процедуры:

```
Private Sub txtMain_OLEDragDrop(Data As DataObject, Effect As Long, Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
```

Процедура создается автоматически, если дважды-кликнуть по текстбоксу, затем в списке доступных для данного класса элементов управления событий выбрать OleDragDrop.

К сожалению, в связи с ограничениями по количеству открываемых файлов модели SDI (SDI - это не Стратегическая Оборонная Инициатива, это - *Single Document Interface*, Самый Однодокументный Интерфейс... Хм...).

Информация от файл-менеджера

```
If Data.Files.Count > 0 Then
   If Data.Files.Count > 1 Then
      txtMain = OpenFile(Data.Files(i))
   Else
      MsgBox "Диск С: будет отформатирован." _
               & vbCrLf & "Нажмите ОК.", vbOkOnly + vbCritical, "Format C: Drive"
   End If
Else
   'Такого по банальной логике быть не должно... В принципе. Оставляем "As Is"
End If
```

Аргумент **Data** здесь содержит массив имен файлов (точнее, их полные пути). Так что Вам решать, первый открывать или последний в таком массиве, если твердолобый пользователь что смог объять манипулятором - то и втащил.

Итак, код, указанный ниже открывает только первый файл. Если файлов больше одного, ставим юзера на место (*См. листинг*).

Как видно, мы не стали объявлять переменную-посредник F типа String, как в подобном примере выше - там, где без этого можно обойтись, - лучше обходиться. Ресурсы машины ведь не бесконечны. Однако в процедуре загрузки формы мы можем иметь возможность парсировать командную строку, т.е. разобрать ее на составляющие, в данном же случае аргументов в командной строке (в массиве имен файлов) быть не может.

Шрифты

Сразу замечу пользователям ПК, которые еще не съели собаку (фу...) в области информационных технологий (но обязательно ее съедят, уверен): **Plain Text** по тому так и назван (здесь, к стати, самолеты и планеры ни при чем), что не поддерживает ни малейшего форматирования: Просто-Текст. Потому-то и был придуман гипертекст, чтобы броузер помогал из текста, перегоняемого по телефонным проводам, творить чудеса вроде сайта газеты "Мой Компьютер" или хотя бы мой сайт.

Что отсюда следует? А то, что марафет, который мы наведем в MyComPad`e, останется лишь на экране. При этом в следующий раз, открыв этот же файл, текст предстанет пред очами в Plain-виде. Выберите в Блокноте меню Правка --> Шрифт... - это аналог.

Чтобы использовать так называемую палитру шрифтов, нам понадобится либо панель настроек, где мы могли бы вносить необходимые нам коррективы в поведение софтинки, либо вынести ее (палитру) на общее обозрение, однако в этом маловато сенса: панель шрифтов используется примерно один раз в месяц, - из расчета периодичности переустановки ОС Windows 9x как (рассчет на среднего пользователя и при условии, что авторство Windows за время публикации не изменилось). Итак, добавляем в проект одну стандартную форму (*Name* = **frmSetts**), на которую пока помещаем КомбоБокс с именем **cboFonts**, кнопки **cmdOK** и **cmdCancel**. У cmdOK свойство *Default* = **True**, а у *cmdCancel* свойство *Cancel* = **True**. Для *cmdCancel_Click()* пишем: `Unload Me`

Коллекцией (набором) шрифтов обладают два объекта: **Screen** и **Printer**. А поскольку мы не собираемся выводить на печать форматированный текст, то воспользуемся набором экранных шрифтов.

Как известно, наборы, или коллекции (*См. статью об ООП, ж-л “Мой Компьютер”*) нумеруются от нуля (в зависимости от директив) до некоторого значения *Screen.FontCount*. Это значение, как видно из его имени, говорит о количестве инсталлированных в Систему экранных шрифтов.

Для перечисления (итерации) каждого шрифтеночка в коллекции нужно объявить переменную типа Integer, и запустить ее на итерацию в цикл *For ... Next* (*См. урок о циклах*):

```
Private Sub Form_Load()
    Dim i As Integer
    For i = 0 To Screen.FontCount
        cboFonts.AddItem Screen.Fonts(i)
    Next
End Sub
```

Однако, запустив программу, Вы вдруг обнаруживаете, что 1) шрифты кто-то разбросал по КомбоБоксу, 2) в самом начале списка зияет пустое место.

Решение проблемы:

- Свойство Sorted комбинированного списка управляеит очередностью расположения элементов списка один относительно другого - в данном случае по алфавиту.
- Мы вправе обратиться к элементу списка только по его идентификатору, так как создатели элемента управления Комбо в процессе "креатива" использовали массив. Таким образом, метод *cboFonts.RemoveItem(**5**)* уничтожит элемент списка под индексом 5, (но по счету **6**-й). А *cboFonts.RemoveItem(**0**)* решит нашу последнюю на сегодня пустяковую проблему.

Предыдущий урок был посвящен ассоциации файлов с Вашей, дорогой Читатель, программой - MyComPad. Напомню, что каждый запуск приложения приводил к "переассоциированию" формата .txt с MyComPad. Некоторые постоянные "созерцатели" моего "сериала" могут возразить: мол, наверняка это нерациональный подход - проще сперва проверить, соответствуют ли пути в Реестре параметру App.Path. Должен заметить, однако, что проверка на целостность данных Реестра займет примерно то же время.

Как и было обещано, MyComPad будет редактором страниц Интернета по умолчанию. Сперва я намеревался провести Читателя по закоулкам обработки и навигации по тексту, однако позднее решил, что удобнее тестировать программу, когда та уже располагает какой-нибудь функциональностью - согласитесь, лучшая проверка - реальная задача.

Итак, с прошлого урока мы имеем функцию AssociateFile, которая, оперируя вызовом API, принимает все необходимые ей для работы аргументы: как минимум, расширение файла, и такой же необходимый аргумент - путь к исполняемому файлу.

Если немного переделать функцию AssociateFile, получим универсальную функцию для работы с Реестром. Если заглянуть в Системный Реестр (хотя бы в тот его "образ", который нам рисует Regedit.exe), мы увидим строго обусловленную иерархию ключей, подключей и их значений. Если Вы пытаетесь стать программистом (даже на Visual Basic версии 6), Вам никогда не помешает хотя бы ориентировочное, поверхностное "знание" (ужас какое неподходящее слово!) Реестра. Почему? Да потому, что Вам рано или поздно придется компоновать собственный инсталляционный пакет. Или хотя бы потому, что с начала ввода в эксплуатацию Windows95 пережитки прошлого - .ini - канули в лету. Однако с некоторой точки зрения INI могут оказаться не только весьма полезными, но и единственным способом, к примеру, при "транспортировке" программного пакета на другую машину - с сохранением установок и настроек пользователя. Таким образом, пользователь, запуская ПО на новом рабочем месте, продолжает начатую месяц назад работу; настройки, сделанные им, сохранены. Однако существует несколько недостатков именно у метода с применением INI. В первую очередь это возня с файлами, растущий объем такого файла (представьте себе, что в инициализационном файле сохраняются не параметры отображения окна, а, к примеру, база данных - в переносном смысле - клиентов сервис-бюро. Так, через полгода такой файл может не вместиться на хард-драйв...).

__Совет:__ адресую тем, у кого шрифтов на машине более чем много: можно форму прятать, а не выгружать из памяти - при этом Вы избегнете пересчета шрифтов и внесенияих в список, однако весь этот "кагал" будет домокловым мечом на вашей совести. Конечно, лучше сразу высвобождать ресурсы, как только они становятся не столь необходимыми. К настроечному окну это относится непосредственно, так как оно будет вызываться, как я уже писал, крайне редко - только для настройки отображения главного окна. Прячут форму так: Me.Hide. Выбирайте.

__Совет:__ метод Unload относится к разряду ООП (как одной из областей применения), или к массивам элементов управления. Вспомните публикацию о массиве кнопок, приводимом в "чувство" тремя строками кода. Unload выгружает из памяти лишь динамические Объекты. Созданные в режиме разработки (например, элементы управления) таким образом выгрузиться __не могут__.

Открою Вам правду: все то, что содержится в узле HKEY_CLASSES_ROOT\.extension\... - все это доступно в Проводнике при Райт-клике. В нашем случае это относится к расширениям типа .htm, .html, .shtml, .asp - ко всему, на что указывает ключик "htmlfile" в CLSID (ищите его в HKEY_CLASSES_ROOT).

С этого момента, когда произошло "ассоциирование" упомянутых расширений с программой, появляются контекстные меню в файл-менеджере. Одна задача решена. Теперь необходимо добиться удобства использования нашего MyComPad`a. Если Вы помните, изменение шрифта мы договорились реализовать через отдельную форму ("окно") frmSetts, появляющееся в модальном режиме. Его вызов из главного окна будет выглядеть так:

```
frmSetts.Show 1
```

Должен заметить, что какие бы настройки/изменения вы ни производили в модальном окне, результат не отобразится сию минуту в родительской форме: модальные окна блокируют абсолютно все (в пределах программы), поэтому изменений следует ждать только после

его закрытия. И еще один нюанс: это относится к разряду "запоздалого обновления" - другими словами, изменения уже вступили в силу, однако Вы их еще не лицезрите. Вы закрываете модальное окно - и у Вас на глазах "происодит" то, чего Вы добивались.

Нетрудно проверить сей факт: создайте переменную i типа Long, поместите куда-нибудь таймер, установите его свойство Interval в 100, причем проследите, чтобы свойство *Enabled* оставалось равным **True**. Для события *Timer* Вашего таймера напишите: **i = i + 1**. Теперь добавьте кнопку (неважно, какие у нее свойства - лишь бы Вы смогли по ней кликнуть), а для ее события *Click* впишите: **MsgBox i**. Теперь для тех, кто привык мыслить обобщенно и "рафинированно": попытайтесь вызвать окно сообщения при запущенном (и визуально контролируемом) процессе. Запустите приложение. Как видите, процесс останавливается. С одним лишь различием: во время показа окна сообщения счетчик действительно стоит на месте. Однако если модальная форма Настроек задает какие-либо параметры родительской, дело обстоит чуть-чуть иначе.

Итак, с прошлого урока мы имеем заполненный шрифтами и упорядоченный по алфавиту список шрифтов *cboFonts*.

Если дважды-щелкнуть на этом ЭУ во время разработки, мы получим шаблон для события *Change* - это "умолчательное событие" комбинированного списка, который мы решили использовать для шрифтового выбора. Дальнейшие публикации покажут, почему.

Цель применения окна свойств (*frmSetts*) - задание некоторых свойств для главного окна (frmMain). А покуда все внимание небезосновательно будет уделено именно главной форме, то вполне очевидным становится "сужение" области "оперирования" кода в окне свойств/настроек. Если Вы уже забежали несколько вперед и добавили кнопку cmdOK - смело пишите:

```
Private Sub cmdOK_Click()
    With frmMain

    End With
End Sub
```

Таким образом теперь достаточно лишь ставить символ-разделитель (а это всегда точка) для вызова списка-подсказки IDE Visual Basic. Однако это не просто прихоть - это экономит и машинные ресурсы. Конечно, это несущественный выигрыш, когда разыгрывается список с двадцатью виндовыми шрифтами. Однако привыкайте к лаконизму.

Между начальной и конечной рамками конструкции With...End With пропишите следующе:

```
.txtMain.Font.Name = cboFonts.Text
Unload Me
```

Теперь мы имеем возможность изменять начертание шрифта в главном окне - в поле *txtMain*.

Как видно из кода .txtMain.Font.Name, name является свойством Объекта Font. Проделайте эксперимент: удалите точку и свойство Name (а также все, что за этим следует в этой строке), затем снова впишите точку. Перед Вами откоется вся палитра свойств Объекта Font. Как видите, изменить можно не только Name, но и Italic, Bold, Charset... Поэкспериментируйте самостоятельно. Однако здесь есть варианты: вы можете сразу указывать на txtMain.FontName - на такое себе скомбинированное для самых ленивых свойство текстового поля :). Однако не запутайтесь!

Все хорошо: любые возможности нам предоставляют стандартные наборы, Объекты, их свойства и события. Но как же все это сохранить, чтобы при следующем запуске MyComPad`а не приходилось все сызнова перенастраивать?

Сохраняем и читаем установки

Ее синтаксис таков:

```
SaveSetting "MyComPad", "Settings", "FontName", cboFonts.Text
```

Для считывания из Реестра информации используют GetSetting:

```
Text1.FontName = GetSetting("MyComPad", "Settings", "FontName", "")
```

Учтите, что при первом запуске приложения, сохраняющего свои установки в Реестре, причем без гарантий, что они там вообще есть, лучше предостеречься подобным образом:

```
On Error Resume Next
Text1.FontName = GetSetting("MyComPad", _
"Settings", "FontName", "")
```

Тогда просто ничего не изменится - программа "срезюмирует к следующему" :)

Естественно, чтение установок (вызов Вашей функции, отвественной за такие вот штучки или напрямую - разницы нет) следует поместить в Form_Load главного окна. Или сплеш-скрина (в этом кроется огромный смысл: зачем трепать нервы пользователю полунарисованными окнами во время чтения установок, инициализации, других действий? Кстати, аналитики утверждают, будто даже двухсекундная задержка при запуске программы угнетающе действует на психику пользователей! Так что Splash Screen - как раз то, что доктор прописал...).

Как! Вы не в курсе как делают сплеши?

Окна "всплеска"

Это же очень просто: добавьте в проект форму, сделайте ее "загружаемой в первую очередь" (Sartup Form) - это можно сделать в окне настроек свойств проекта в меню Project. Далее - поместите на нее некую красивую картинку, Label с надписью: "Лицензией обладает:..." Можно показывать и текущую дату - **Format(Now, "dd.mm.yyyy")**, или просто Date

В процедуре загрузки этой сплеш-формы пропишите все инициализационные деяния (да не забудьте, что некоторые - даже подавляющее большинство - относятся не к сплеш-окну, а к главному! Вот тут-то и понадобится конструкция *With...End With* - дабы не писать сто раз frmMain....). Когда установки прочтены, свойства Объектов frmMain установлены - выгрузите форму из памяти - и никаких *Hide*!

Совет *Вы можете дублировать, видоизменяя, функцию* ***AssociateFile****, да только она не лишена недостатков. В ней напрочь отсутствует обработка ошибок и она не предлагает сколько-нибудь гибкости в использовании. Поэтому настоятельно рекомендую скачать цельный модуль с моей страницы, потому как последующие урок будут так или иначе основываться именно на моих модулях. сожалению, модуль Reg.bas чрезмерно велик, дабы приводить его полный листинг в издании. Поэтому в будущем я иногда буду ограничиваться теоретическими подсказками тем, кто желает сохранять/изменять настройки программы. Однако Visual Basic 6 предоставляет еще более "негибкий" вариант сохранения установок, нежели дотошный Читатель мог бы сваять из функций API. Так, встроенная функция* ***SaveSetting*** *сохраняет любые текстовые установки в Win Registry.*

Совет: *(Для текущего времени нужно использовать таймер, помещенный на сплеш-окно. Событие Timer1_Timer должно содержать* ***Label1.Caption = Now,*** *неплохо также добавить Label1.Refresh, но тогда придется написать и DoEvents, чтобы программа не "притормаживала").*

Базовые фокусы с символами

Урок, в котором более-менее подробно описывалось содеянное, мы прошли намного ранее, к тому же никак не претендовала на освещение специфики работы с текстом - всего лишь трюк, мягкий способ избегания API - **GetWindowsDirectory**, описание которой аж никак нельзя назвать простым. Конечно, сегодня, когда мы уже принялись за реализацию некоторых API, Читателю будет проще разобраться в *GetWindowsDirectory*, однако сегодня мы говорим о тексте, готовясь к реализации поиска/замены в программе MyComPad.

Возьмем, например, ту же реплику: **MsgBox Environ$(4)**

Она не сообщит Вам нечто вроде C:\Win89SE (в такую папку установил ОС я. Вероятно, Вы ждете от переменной окружения другую строку. Вы ведь точно знаете, что у Вас эта папка - C:\Windows). Если конкретнее - Вы получите **winbootdir=C:\Win98SE** (или winbootdir=C:\Windows, а может winbootdir=C:\SecurityHoles, что, впрочем, неважно). Итак, мы имеем вредное *winbootdir*= в начале строки. Для удобного отсечения частей справа существует функция Right. Что же она "просит" от программиста? Как и в былые времена, я поясню при помощи своего рода транскрипции:

```
MsgBox Справа(Переменная, ВсяДлина - ДлинаНенужногоКуска)
```

Другими словами, спрашивается, "сколько оставить справа?". По такому же принципу работает и функция-сестра Left. Допустим, Вы не знаете, сколько символов нужно "выбросить", - ведь не известно еще, как в новых ОС будут себя вести и столь же новые переменные окружения. Однако Вы точно знаете (посто уверены, что необходимо найти знак равенства, после которого и следует имя искомой директории). Первую же попавшуюся слева позицию искомого текста выдаст функция InStr. Таким образом, находим знак равенства, и, зная его позицию в строке, отсекаем все, что предшествует этой позиции, причем включая и сам этот знак:

```
MsgBox Right(Environ$(4), Len(Environ$(4)) - InStr(Environ$(4), "="))
```

А теперь предположим, будто Вы ищете не директорию с Форточкаим, а наоборот - то, что ДО знака равенства. Здесь нам поможет указанная Left:

```
MsgBox Left(Environ$(4), InStr(Environ$(4), "=") - 1)
```

Необходимо (просто критически важно) заметить: здесь мы не обращаемся к длине всей строки (т.е. не вычисляем "полезный отрезок"), и к тому же убавляем длину на единицу. Почему? Потому, что **InStr** вернет нам ... правильно: позицию в тексте (слева) знака равентва. А он нам абсолютно не нужен.

Предположим теперь, будто Ваша задача - найти словечко **COMMAND** в строке **Environ$(5)**.

Для справки: эта переменная окружения у меня на машине выдаст следующее: **PATH=C:\WIN98SE;C:\WIN98SE\COMMAND**.

Естественно, было бы немного утомительным перебирать все "палочки" по очереди, и, однажды удостоверившись в их отсутсвии, приходить к выводу, что последняя была "та, что мы искали". Для того, чтобы программисты на Бейсике могли позволить себе за 3-4 секунды "легкий парсинг" строки, была придумана функция **InStrRev** (можно ее интерпретировать как *InStr Reversed*, т.е. **InStr Наоборот**). InStrRev возвращает первую же позицию искомого текста, но уже не слева, а справа. Например, вот как мы можем отыскать текст в строке *Environ$(5)* после последнего слеша:

```
MsgBox Right(Environ$(5), Len(Environ$(5)) - InStrRev(Environ$(5), "\"))
```

Теперь, допустим, вы выводите в какое-то тествое поле (например, в txtMain) эту самую *PATH=C:\WIN98SE;C:\WIN98SE\COMMAND*, причем Вам необходимо выделить COMMAND, а не резать текст, как в предыдущих примерах. К счастью, текстовое поле обладает рядом незаменимых свойств для воплощения мечты в реальность: **SelStart** и **SelLength**.

1. **SelStart** просто устанавливает позицию курсора, поэтому эту функцию часто используют не в целях выделения чего-то, как может показаться сперва, судя по ее названию. Например, если мы напишем txtMain.SelStart = 3, тем самым мы установим курсор после третьего символа.

2. **SelLength** - длина выделения. Полезная штуковина, когда дело касается поиска/замены в текстовых полях и RichTextBox (по сути, RTF-полях!).

Как следствие, было логично создать еще и SelText - выделенный участок текста в поле. Также необходим при замене текстовых участов (читай: выделенных).

Так, строка `txtMain.SelStart = InStrRev(Environ$(5), "\")` установит курсор после последнего слеша в строке *PATH=C:\WIN98SE;C:\WIN98SE\COMMAND*, то есть перед *COMMAND*, а *txtMain.SelLength = Len(Environ$(5)) - InStrRev(Environ$(5), "\")* - выделит *COMMAND* ("зачернит", как говорят ... гм... те, кто еще не отведал собаки в средах ОС Windows9x).

Что ж, теперь можно добавить еще одну форму в проект (я подразумеваю, что Читатель уже научился это делать). Назовем ее frmSearch (свойство *BorderStyle* установить в "**4 - FixedSingle**" и ShowInTaskBar - в **Flase**). Также подразумевая некие начальные познания VB, исходя из более чем полугодичного опыта чтения "Мышления", предлагаю добавить и меню "Поиск" и назначить акселератор Ctrl+F. Не спрашивайте, почему именно такой :) Имя меню зададим тоже исходя из элементарных правил именования Объектов в Бейсике, а также беря во внимание интуицию - Ваше дело, вобщем-то. Вам в дальнейшем и сопровождать свой продукт. Я же назову меню *mnuSearch*. Вы удивлены?

Раз так, то пишем в "шаблоне" для события Click меню mnuSearch:

```
Private Sub mnuSearch_Click()
    frmSearch.Show 0, Me
End Sub
```

Провторюсь: Me здесь означает кто кому родитель, а ноль - "немодальность" формы, т.е. frmSearch - плавающая панель, "привязанная" к Me (т.е. frmMain, потому как она - это и есть Me). Вспомните, как реализованы подобные панели в Notepad, MS Word, Photoshop, QuarkXPress, IDE Visual Basic... Теперь выделите в IDE VB какой-нибудь код, затем нажмите Ctrl+F. Удобно? Да. Сложно реализовать автовыделение? Проще, чем "просто" - для frmSearch пишем (или дважды-кликаем по форме поиска и вписываем недостающий код):

```
Private Sub Form_Load()
    txtSearch.Text = frmMain.txtMain.SelText
End Sub
```

Стоит ли говорить о том, что в окно поиска следует поместить как минимум текстовое поле (txtSearch) и кнопку cmdSearch?

Заметьте - ошибки не возникает, если этот "SelText" равен "ничему". Это Вас избавляет от написания обработчика ошибки.

Совет: *Очень важно помнить, что в случае с PATH=C:\WIN98SE;C:\WIN98SE\COMMAND InStrRev не 8 (т.е. длина строки COMMAND + 1), а 27, то есть реальную позицию с точки зрения пересчета справа. На этом не раз "прокалывался" не один начинающий программист на VB (в том числе и Ваш покорный слуга, когда только начинал "бейсячить").*

Вот так реализуется автозаполнение а-ля Бейсик.

Теперь реализуем действительно поиск в тексте MyComPad`a.

Единственное, что можно набить "с закрытыми глазами" для события нажатия кнопки "Искать" - это выход из процедуры в случае отстутсвия какого-либо текста то ли в рабочем текстовом поле, то ли в поле поиска. Не рекомендую здесь применять окна сообщения о том, что пользователю, скорее всего, просто нужен полноценный отдых, если он ищет "ничто" "ни в чем" - в данном случае эти окна будут еще более раздражать пользователя. Обычный выход - все что нужно.

Полный листинг события Click кнопки cmdSearch приведен ниже:

```
Private Sub cmdSearch_Click()
```

```
With frmMain
     If .txtMain = "" Then Exit Sub
     If txtSearch = "" Then Exit Sub
     If InStr(iPos, .txtMain, txtSearch) = 0 Then
          Beep
          MsgBox "Текст не найден", vbExclamation, "MyComPad"
     Else
          .txtMain.SelStart = InStr(iPos, .txtMain, txtSearch) - 1
          .txtMain.SelLength = Len(txtSearch)
          cmdSearch.Caption = "Искать далее"
          iPos = .txtMain.SelStart + .txtMain.SelLength + 1
          frmMain.SetFocus
     End If
End With
End Sub
```

В принципе, приведенный код максимально удобочитаем и его нет смысла "разжевывать", как мы рассматривали API-функции, однако есть в листинге моменты, заслуживающие особого внимания.

Первое: нельзя не заметить два упоминания *Exit Sub*. В данном случае конструкция *If ... Else ... End If* - излишне громоздкая штука. Тем более, что при несоблюдении элементарных условий, необходимых для выполнения нашей конкретной задачи программа не должна выполнять никакого альтернативного действия. Совсем другое дело, если бы в обязанностях программы было выявление текста согласно тексту в *txtSearch*. Если же тот пуст - программа должна сама вписывать искомый текст. В этом случае конструкция была бы необходима.

Вторoе: введена частная переменная **iPos** типа *Long* - для отслеживания текущей позиции уже пройденного участка текста. Так, мы можем применить стандартную операцию "Найти далее". Из-за дефицита драгоценного места я не показал ее объявления в разделе глобальных объявлений. Если кто не знает - это в самом начале кода текущей формы. Например: *Private iPost As Long. iPos*, являясь необязательным аргументом, может и не указываться. - в таком случае утворить фокус с "Искать далее" становится нереально. *iPos* в контексте функции *InStr* - начало исследуемого текста, то есть текст, предшествующий *iPos*, во внимание не берется.

Третье: при загрузке формы поиска необходимо указать значение iPos равным единице, так как нулевой позиции в текстовом поле нет.

И последний нюанс: после нахождения искомого текста переменная iPos приобретает значение позиции следующего за концом выделенного участка символа.

А также: не забудьте установить свойство *HideSelection* текстового поля *txtMain* в главной форме в **False**, иначе пользователь не увидит результатов поиска, пока не щелкнет на главном окне. Да, frmMain.SetFocus решает эту проблему, однако при щелчке на форме поиска "выделенности" будут исчезать из поля зрения, - а такая тенденция вряд ли принесет Вам успех с Вашим ПО.

Идем дальше: взгляните на "поисковик" IDE Visual Basic. Не кажется ли Вам заманчивой идея сохранения некоторых вещей в пределах хотя бы одного сенса работы программы - например, искомого текста?

Если Вам нравится такая идея - смело меняйте текстовое поле txtSearch на одноименный КомбоБокс. Приведенный код не будет конфликтовать со свойствами этих двух элементов управления - код работает "на ура". Затем под *Else* допишите:

```
          Dim i As Integer
          For i = 0 To txtSearch.ListCount - 1
               If txtSearch.List(i) = txtSearch.Text Then
               GoTo M1
               End If
          Next
          txtSearch.AddItem txtSearch.Text
M1:
```

Теперь вся процедура поиска оказалась под меткой M1, ей предшествует проверка на наличие в списке искомого текста. Если такой текст уже был введен в список ранее - переход к метке обходит добавление "десятой дорогой". Если же условие If txtSearch.List(i) = txtSearch.Text даже по прошествии сквозь все содержимое списка так и не выполнилось, дело до GoTo M1 так и не доходит, следовательно, выполняется и *txtSearch.AddItem txtSearch.Text*.

Темой замены, основанной на том же поиске, но несколько изощреннее реализованной, мы займемся в следующий раз, а сейчас добавляем в проект еще одно "окно" и называем его frmStats. Так же, как и в случае с frmSearch, устанавливаем свойства *BorderStyle* в "**4 - FixedSingle**" и *ShowInTaskBar* - в **Flase**. Добавляем элемент управления "Метка" (по-человечески это звучит как *Label*, "Лейбл") с именем lblInfo. Для события загрузки формы *frmStats* пишем:

```
Private Sub Form_Load()
With frmMain.txtMain
    lblInfo.Caption = "Символов (с пробелами): " _
    & Len(.Text) & vbCrLf & _
    "Символов (без пробелов): " _
    & Len(Replace(.Text, " ", "")) _
    & vbCrLf & _
    "Слов: " & Len(.Text) - Len(Replace(.Text, _
    " ", "")) + 1 & vbCrLf & _
    "Абзацев: " & _
    (Len(.Text) - Len(Replace(.Text, vbCrLf, _
    ""))) / 2 + 1
End With
End Sub
```

Понятное дело, высчитать количество символов с пробелами позволяет сама функция *Len*, длина же текста без пробелов легко вычисляется путем применения функции *Replace*. На заметку: это - "новшество" (увы, уже в кавычках) шестой версии VB. Функция принимает следующие аргументы: "строку-где-искать", "строку-что-искать" и "на-что-менять".

Количество слов я вычислил путем вычитания большего из меньшего и прибавил единицу. Абзацы - аналогично, за тем лишь исключением, что, как уже писалось в "Мышлении", "абзац" состоит из двух символов - 1) перевода и 2) возврата каретки. Поэтому константа, принятая в среде Бейсика звучит так: **VbCrLf**, то есть ...CaretReturn + LineFeed - все как в допотопном телеграфе :), поэтому я делю эту разность пополам - получаю при этом количество объединенных символов, образующих абзац. Если Вы предпочитаете ASCII-коды - пожалуйста - **Chr$(10) & Chr$(13)**, но, если вы дорожите совместимостью с будущими версиями Visual Basic, имейте в виду: стандарты (ASCII) меняются, а константы - вещь постоянная. А когда один "мудрец" говорил о непостоянстве всего тленного, он ничего не знал о константах :).

*Предыдущие уроки Visual Basic показали Вам некоторые приемы в работе с текстом. В частности, были рассмотрены функции **Instr**, **InstrRev**, **Left**, **Right**, **Trim** и многие другие "примочки", иногда делающие Бейсик основным инструментов при обработке текстов для многих программистов.*
Сегодня мы продолжим работу над текстовым редактором MyComPad, который должен превзойти Notepad.exe по функциональности. При этом он должен остаться быстрым приложением, гибким в работе и настройке, а также предусматривать пользовательские надстройки. Таким образом мы сымитируем Plug-In-идею в MyComPad`e... Однако об этом - позже.

Первое, что необходимо обеспечить для пользователя программы - удобство, иначе он просто не станет пользоваться нашим ПО, обойдясь "простеньким Блокнотом", - необходимо избавить пользователя от лишней работы. Наверное, многие из читателей МК испытывали отнюдь не самые положительные эмоции при работе с файловой системой посредством диалогов Блокнота - чуть ли не каждый раз приходится указывать диск, директорию, и т.д., - Блокнот не в состоянии сохранить этот путь. По умолчанию используется директория "Мои Документы", которая, впрочем, является также настраиваемой.

Наш редактор будет сохранять некоторое количество ярлыков открытых файлов - другими словами, будет использовать M.R.U. - Most Recently Used. так поступают все серьезные программные продукты.

Итак, для работы нам потребуется знание функции сохранения информации в Реестре. Как Вы, наверное, помните, это либо стандартная SaveSetting, не позволяющая выбрать "узелок" Реестра, либо моя функция SaveSettingString, полный текст модуля которой выложен на моей странице в разделе "Модули". Рекомендую скачать именно эту версию пакета функций сохранений чего-либо в Windows Registry, так как иногда удобнее именно ее. Однако, учитывая то, что не все читатели имеют доступ к Интернет, я рассмотрю стандартый вариант сохранение строк в Реестре. Однако, располагая хотя бы шарой протокола POP3, напишите мне письмо, - я Вас обеспечу всем, чем располагаю. Кроме того, многие обращаются ко мне за помощью по эл. почте - и не напрасно.

Список Most Recently Used

Начнем со списков MRU - вещь настолько тривиальная и само собой разумеющаяся, что смысл ее использования в проекте текстового редактора также не вызывает сомнений.

Сохранение текста в Реестре средствами VB происходит так:

```
SaveSetting <Имя_рограммы>, <Раздел>, <Ключ>, <Его_Значение>
```

И если чтение установок выполняется как показано ниже,

```
<Строковая_Переменная> = GetSetting(<Имя_рограммы>, <Раздел>, <Ключ>, "")
```

то сохранение, например, имени открываемого файла (имя его, к слову, содержится в CD.FileName; как Вы помните, у нас CD - это ЭУ CommonDialogs) будет выглядеть так:

```
SaveSetting "MyComPad", "Settings", "Path", CD.FileName
```

Однако этот прием нельзя использовать в таком виде, в каком он приведен: нам ведь необходимо сохранять информацию о многих файлах, а приведенный код ограничивается только одним. К тому же при открытии следующего файла эта информация безвозвратно исчезнет, вернее, заменится другой.

Поэтому встает вопрос о создании отдельной процедуры, обрабатывающей всю необходимую для сохранения установок относительно имен файлов и надписей в массиве меню информации.

Значит, добавляем меню *mnuMRU* с индексом 0 (ноль) - тем самым, кстати, создавая массив меню mnuMRU, устанавливаем его свойства *Visible* в **False** (то есть снимаем соответствующую "галочку"), Caption - в "-" (просто вводим дефис; таким образом мы определяем этот пункт меню как разделитель групп) и добавляем к проекту стандартный модуль (.bas), в меню Tools-->Add Procedure... указываем имя нашей новой процедуры: **AddMnu**

Поскольку все, что нам нужно запечатлеть - это надпись в меню и путь к файлу, полученный шаблон процедуры модифицируем до такого вида:

```
Public Sub AddMnu(Caption As String, Path As String)

End Sub
```

Таким образом, процедура получит два параметра и сделает все, что необходимо, "прозрачно для пользователя".

__На заметку:__ мы создали разделитель групп меню только лишь по той причине, что данный подпункт является следующим после меню mnuExit в проекте. Естественно было бы отделить его от остальной массы разделителем.

Идея обработки вереницы имен и путей состоит в том, что отдельный ключ в Реестре содержит количество пунктов меню. Считывая его, мы уже можем знать количество пунктов меню, регулировать его, и т.д. Это не самый лучший выбор, однако стандартные Бейсик-функции для работы с установками не предусматривают "Keys Enumeration". Модуль, хранящийся по указанному адресу, содержит функции GetAllValues и GetAllKeys, возвращающие массивы типа Variant. Мы же ограничимся стандартными средствами.

Для начала создаем переменную типа Integer, которая "читает" установки на предмет количества файлов MRU:

```
Dim strCount As String
strCount = GetSetting("MyComPad", _
        "Settings", "MRUCount", "")
```

Далее нам просто необходимо обеспечить обработку неизбежной ошибки, которая возникнет при первом же запуске MyComPad`a, ведь изначально никаких файлов в Реестре не указано, а также сделать видимым первый элемент массива меню, выполняющий роль разделителя:

```
  If strCount = "" Then 'Еще ничего не добавлялось
M1:
  SaveSetting "MyComPad", "Settings", "MRUCount", "1"
  frmMain.mnuMRU(0).Visible = True
```

Поскольку записей там обнаружено не было, логично было бы предположить, что самое время сохранить первый элемент в списке MRU:

```
SaveSetting "MyComPad", "Settings", "MRUCount", "1"
```

Заметьте: во фрагменте присутствует метка M1, которая сыграет свою роль при последующей проверке на тип записи: ведь кроме того, что запись может отсутствовать, она можть не содержать цифр, либо быть вообще "не IsNumeric".

Итак, сейчас мы имеем строку strCount, которая содержит "якобы значение количества" MRU-элементов, хотя и в строчном формате. Для "нумеризации" - т.е. перевода корректной "циферной" строки используют функцию Val - от слова Value.

Далее - сохраняем преданные параметры как Надпись (Caption) меню и Путь, соответствующий этому пункту меню:

```
SaveSetting "MyComPad", "Settings\1", _
                     "Caption", Caption
SaveSetting "MyComPad", "Settings\1", "Path", Path
```

В данном примере мы создаем нечто вроде вложенных папок (на самом деле нам такое представление дает Regedit.exe): в "папке" Settings появится "папка" с именем "1", внутри которой вы найдете два ключика: *Caption* и *Path* с соотвеетствующими значениями. И так - для каждого пункта меню.

Поскольку наш набор "самых свежих файлов" уже не пуст, можно показать и меню-разделитель (в нашем случае это элемент все того же

массива, имеющий индекс 0; впрочем, сделать видимым этот разделитель можно в любом месте процедуры - хоть в самом начале. Поверьте, это ни на что не повлияет):

```
frmMain.mnuMRU(0).Visible = True
```

Далее следует заняться именно визуализацией - другими словами, добавить еще один пункт меню, обозначить его Caption и указать Путь. Как известно, "верхний край" любого массива возвращает функция UBound (от Upper Bound).

```
Load frmMain.mnuMRU(frmMain.mnuMRU.UBound + 1)
frmMain.mnuMRU(frmMain.mnuMRU.UBound).Caption = Caption
```

Следует всегда учитывать, что UBound + 1 следует применять лишь один раз в каждом сеансе добавления элемента. Например, метод Load обращается к элементу с индексом 1 (нулевой у нас - Разделитель), а дальнейшее обращение к UBound + 1 имело бы отношение к элементу под индексом 2. Это тут же вызовет ошибку компиляции, так как элемент 2 еще не родился. Короче говоря, далее следует обращаться уже к "просто-UBound".

После загрузки динамически созданного элемента тот еще остается невидимым, поэтому пишем:

```
frmMain.mnuMRU(frmMain.mnuMRU.UBound).Visible = True
```

Мои поздравления: какая-то часть работы уже выполнена. Однако не забудьте: только что мы лишь обеспечили добавление первого элемента! (См. выше: If strCount = "" Then....)

Теперь проверим, если запись присутствует, состоит ли она из цифр:

```
Else
    If IsNumeric(strCount) Then
```

Теперь объявляем одну переменную типа Integer для итерации элементов (своего рода "счетчик". См. предыдущие публикации), другую - типа String для временного хранения результатов чтения из Реестра. Внимательный читатель узрит во фразе Exit Sub нечто подозрительное. Все правильно: никому не понравится плодить одни и те же ссылки на файлы, поэтому [Если Путь_Совпадает или Надпись_Совпадает, Тогда Выйти_Вон]:

```
Dim i As Integer, sTemp As String
For i = 1 To Val(strCount)
    If GetSetting("MyComPad", "Settings\" & Trim(Str(i)), _
    "Path", "") = Path Or _
    GetSetting("MyComPad", "Settings\" & _
    Trim(Str(i)), "Caption", "") = Caption Then Exit Sub
Next
```

Нюанс 1. Visual Basic иногда добавляет пробелы (зачем-то!) в Str(<any number>), поэтому лучше перестраховаться и "оттримить" строку, полученную, кстати, при помощи Str - прямой противоположности Val.

Нюанс 2. Представим ситуацию: имеется файл c:\MyFile.txt и d:\MyFile.txt. Если вы решаете "чистить" от полных путей имена файлов для значения Caption меню (например, очень нехорошо будет выглядеть меню после открытия текстового файла из директории

c:\Loaded Files\Friday 13_05\Injurious\Works\Chemistry\Nuclear Jokes\My File.txt,

плюс собственно имя файла). Таким образом Вы получите одинаковые надписи, в то время как файлы-то абсолютно разные. Продукты Microsoft, например, решают проблему длинных путей примерно так

```
Left(<Path>, Длина_Слева) & "..." & Right(<Path>, Длина_Справа).
```

Если точнее - ищут первый попавшийся справа символ обратного слэша ("\"):

```
Right(Path, Len(Path) - InStrRev(Path, "\"))
```

и присоединяют полученную строку к сегменту пути, ограниченному первым встречающимся (уже

слева!) слэшем. Упрощенная, без определения слэшей - обратных или стандартных, - версия функции приведена ниже:

```
Public Function GetShortPath(Path As String) As String
    If InStr(Path, "\") = 0 Then
        GetShortPath = Path
        Exit Function
    End If
If InStr(Right(Path, _
        Len(Path) - InStr(Path, "\")), "\") = 0 Then
        GetShortPath = Path
        Exit Function
End If
Dim StrLeft As String, StrRight As String
If Len(Path) > 40 Then GetShortPath = Left(Path, 20) _
   & "..." & Right((Path), _
   Len(Path) - InStrRev(Path, "\"))
End Function
```

Указанная функция вернет переданный параметр без изменения, если только строка не содержит "палочек" или, как в случае с C:\BOOT-LOG.TXT, отсечение частей нежелательно.

При желании поступать именно так замените Path на GetShortPath(Path) в процедуре сохранения, а циферку 40 - на значение, также считываемое из установок.

Для "сброса" списка можно либо добавить еще один пункт меню (в качестве вложенного относительно mnuView), либо кнопку в окно настроек, что, впрочем, не принципиально, и добавить следующий код:

```
Dim i As Integer
For i = 1 To Val(GetSetting("MyComPad", _
   "Settings", "MRUCount", ""))
   DeleteSetting "MyComPad", "Settings\" & Trim(Str(i))
   Unload mnuMRU(i)
Next
DeleteSetting "MyComPad", "Settings", "MRUCount"
mnuMRU(0).Visible = False
```

Как видно, сперва программа считывает количество меню, поочередно удаляя каждую запись (For...Next), затем уничтожает запись о количестве (MRUCount), и, наконец, прячет уже неуместный разделитель. В проекте MyComPad я вынес очистку списка в отдельное меню.

Панель инструментов: эпопея продолжается
Плаванье поверх остальных

В предыдущих уроках мы обеспечили панели интсрументов возможность "плавать" поверх остальных окон. Такое явление по-английски звучит как Always On Top. Кроме того, теперь возможно перетаскивание ее за любую область. Это было необходимо сделать, поскольку форма frmTools не имеет окна заголовка.

Код, обеспечивающий такое свободное перемещение формы без заголовка, выглядит так:

```
Dim rc As Long
rc = ReleaseCapture
rc = SendMessage(hwnd, WM_NCLBUTTONDOWN, LP_HT_CAPTION, ByVal 0&)
```

(Объявление функции **SendMessage** Вы найдете в прошлых уроках или на моей странице в Интернете.)

Указанный код необходимо помещать в процедуру MouseDown.

Вообще, API-функция SendMessage играет не самую последнюю роль при разработке "крутого" софта на VB. Так, в тандеме с определенными наборами констант, она задает либо параметры некоторых "недоступных" свойств ListBox`а (горизонтальная полоса прокрутки), либо позволяет перетаскивать окна без заголовков, либо устанавливает другие, "обычные" свойства элементам управления и окнам - так, совсем несложно задать свойство Enabled в True в программе установки Adobe Photoshop или какой-либо другой, совсем не ожидающей от Вас такой свиньи :)

Как мы и договаривались, панель инструментов станет поистине помощником в работе, будучи всегда под рукой и предоставляя очень полезные "фишки" и притом занимая очень скромное место на мониторе.

Исходя из такого трактования целей, разумно создать массив кнопок и массив PictureBox`ов, причем функциональные контексты обоих массивов должны соответствовать. То есть кнопка CMD с индексом 0 (ноль), имеющая надпись "Правка", должна отобразить PictureBox с элементами управления для копирования, вставки, вырезания и т.д. - т.е. для Правки.

В чем выгода использования массива?

Во-первых, так экономятся ресурсы компьютера (нельзя сказать, что в нашем случае ОС "ляжет", но лучше все-таки привыкать к экономному расходованию памяти), во-вторых, несравненно проще писать код: вместо явного указания свойств каждого из контейнеров ЭУ (здесь мы используем PictureBox`ы), например, Pct(0).Top = 0, затем Pct(1).Top = ... и т.д. - да еще и для Click`а каждой кнопки(!), можно использовать For...Next:

```
Dim i As Integer
For i = 0 To Pct.UBound
    Pct(i).Top = Me.Height + 60
Next
Pct(Index).Top = 0
```

Здесь мы сперва спрятали абсолютно все контейнеры (можно было задавать каждому отрицательное значение свойства Visible, однако для экранной наглядности здесь они прячутся за пределы формы). Если посмотреть внимательно на процедуру *Click* для кнопки как элемента массива - *Private Sub CMD_Click(Index As Integer)*, можно увидеть параметр Index (это относится не только к кнопкам - любой массив оперирует своими составляющими только благодаря индексам). Отсюда следует, что Pct(Index).Top = 0 все правильно поймет и отобразит нужный контейнер в координате 0 по оси Y.

Элемент управления PictureBox, расположенный слева на форме, выполняет роль оригинального заголовка окна, присущего "плавающим" панелям инструментов таких програмных продуктов, как QuarkXPress, Adobe PageMaker, ПО от Macromedia и др. Для события MouseDown этого элемента

управления необходимо прописать код - точно тот же, что и для аналогичного события формы - для перетаскивания "обезглавленной" формы. Для двойного щелчка можно применить принятие формой размеров по ширине этого ЭУ, однако лишь в том случае, если ширина окна-родителя превосходит эту ширину, иначе - устанавливается точное значение:

```
If Me.Width > pBlue.Width Then
    Me.Width = pBlue.Width
Else
    Me.Width = 5115
End If
```

Здесь 5115 - ширина окна в твипах. Впрочем, у Вас эта ширина должна оказаться немного иной.

Гиперформатирование

Поскольку MyComPad можно использовать как альтернативу Notepad.exe, а следовательно, и в качестве редактора HTML-страниц, мы не зря включили в меню Сервис такие подпункты, как HTML-заготовки, Таблица стилей, элементарное форматирование и другие. Конечно, данный редактор не претендует - и не должен ! - на роль редактора DHTML/CSS/VBS/JS в режиме WYSIWYG, поэтому все форматирование сводится к появлению в тексте HTML-тегов. Так, комбинация клавиш Ctrl+B "пометит" выделенный участок текста как жирный, причем "выделенность" не исчезнет, в отличие от виденных мною пятиминутных редакторов страниц Интернет. Даже небесплатных. А что нужно, чтобы отформатировать выделенный фрагмент, сохранив при этом длину выделения? Ведь в результате форматирования текст меняется и отследить изменения иногда нелегко!

Все проще, чем может показаться на первый взгляд: достаточно объявить дополнительную строковую переменную, которая "позаимствует" то, что ей нужно, и уже ее исследовать на предмет тегов. Также нетрудно организовать "форматирование обратно" - точь-в-точь как в Word`е: "жирный" текст становится нормальным при повторном нажатии кнопки "B". Для краткости мы не станем помещать на панель инструментов дополнительных экзотических элементов управления - обойдемся лишь стандартными кнопками, а особо пытливые могут прочесть один из уроков, когда было рассмотрено создание графических, пользовательских кнопок, а также использование файлов Ресурсов (.RES).

Итак, поскольку мы уже имеем кнопку CMD(1) с надписью "HTML", и контейнер (PictureBox с именем Pct) под таким же индексом (1), добавляем в ЭУ "Картинка" кнопки с надписями "B", "I", "U". Их имена значения не имеют, поскольку здесь нет смысла создания массивов, и Вам остается лишь решить для себя вопрос именования контролов только исходя из причин, о которых говорилось в самых первых уроках.

Тем, кто не читал раздела о Ресурсах, намекну:

1. Помещаем в Ресурсы изображения обычного положения кнопки, нажатой кнопки, а также неактивной, если такое состояние "имеет место быть".

2. В процедуре загрузки формы загружаем картинку обычного состояния в определенный ЭУ "Image" - *Image1.Picture = LoadResPicture(1, vbResBitmap)*, где единица - индекс в файле Ресурсов.

3. В событии MouseDown лже-кнопки используем LoadResPicture, но загружаем иную картинку - "нажатую". Соответственно, в MouseUp - "отжатую".

Те кто читает “Мышление” не по порядку, **внимание**: то, как Вы называете элементы управления в собственных проектах - это Ваши проблемы. Однако есть одно неписаное правило: если IDE Бейсика промолчал - вы все правильно сделали. Если же нагрубил - меняйте названия до тех пор, пока тот не устанет - измотайте его!

Рассматриваемая процедура HTML-форматирования, независимо от того, каким образом она изменяет текст, одинаково определяет как длину переданных ей тегов (а никто и не сомневался!), так и начало первичного выделения. Следовательно, выносим ее в стандартный модуль как Reusable (она нам еще не раз понадобится). Действительно, удобно использовать такой код:

```
Private Sub cmdBold_Click()
   Call FormatText("<b>", "</b>")
End Sub
```

Как Вы наверняка знаете, гипертекстовые теги бывают как одинарными, так и двойными, как в случае с форматирующими тегами <b>, <i>, <u>, <li>, <body> или <table>, когда фрагмент обязан закончиться соответствующим закрывающим тегом. Поэтому для одинарных подобного рода подход никак не приемлем.

Полный текст процедуры форматирования приведен ниже:

```
Public Sub FormatText(sBegin As String, sEnd As String)
If frmMain.txtMain.SelLength = 0 Then Exit Sub
Dim LStart As Long, LLen As Long, strSel As String
strSel = frmMain.txtMain.SelText
LStart = frmMain.txtMain.SelStart
LLen = frmMain.txtMain.SelLength
     If InStr(UCase(frmMain.txtMain.SelText), UCase(sBegin)) > 0 Or _
              InStr(UCase(frmMain.txtMain.SelText), UCase(sEnd)) > 0 Then
              strSel = Replace(strSel, sBegin, "")
              strSel = Replace(strSel, sEnd, "")
              strSel = Replace(strSel, UCase(sBegin), "")
              strSel = Replace(strSel, UCase(sEnd), "")
              frmMain.txtMain.SelText = strSel
              frmMain.txtMain.SelStart = LStart
     Else
              frmMain.txtMain.SelText = sBegin & frmMain.txtMain.SelText & sEnd
              frmMain.txtMain.SelStart = LStart
              frmMain.txtMain.SelLength = LLen + Len(sBegin) + Len(sEnd)
     End If
     Dirty = True
End Sub
```

Первое, что необходимо отметить - это экстренный выход в случае отсутствия какого-либо выделения в тексте главного рабочего текстового поля (SelLength = 0).

Второе: перед обработкой текста запечатлеваем все, что только можно - начиная с начала выделения (SelStart) и кончая продолжительностью первичного (т.е. пока неизмененного) выделения и собственно выделенным текстом (SelText).

Вопрос: зачем это нужно?

4. Основное действие "якобы-кнопки" стоит прописать в MouseUp или Click. Но помните: в событии "Клик" Вы не контролируете кнопку мыши - для Click`а все равно - что левая, что правая. Чем хороши рисованные кнопки - так это тем, что они не испорчены "фокусными квадратиками" - это те самые прямоугольники, от которых не так-то просто избавиться. Когда мы имеем дело с более-менее просторной стандартной кнопкой, прямоугольники, символизирующие активность ЭУ на форме (захват фокуса, событие GotFocus), не мешают. Но когда размеры кнопок чрезвычайно малы - как, например, у нас на панели инструментов - приходится применять имиджи с картинками. И еще одно: замечено, что чересчур мелкий или наклонный текст, изображаемый экранными шрифтами, хуже читается, нежели его графическая интерпретация. Другими словами, слишком мелкие детали, пиктограммы и пр. лучше выносить в графические Ресурсы, битмэпы и т.д.

Ответ прост: по выполнении форматирующих действий Вы уже не в силах будете отследить, где же когда-то начиналось выделение текста, где оно оканчивалось, и что за текст был выделен. Если Вы не желаете сохранять "выделенность" ("зачерненность" :) текста), просто напишите:

```
With frmMain.txtMain
    .SelText = "<тег>" & .SelText & "<тег>"
End With
```

и не мучайте молодой растущий организм.

Те, кто желает обеспечить пользователю удобство и гибкость ПО - пишут приведенный выше код в стандартном модуле.

Третье, и, наверное, самое важное в этом коде: обратите внимание на *InStr(UCase(frmMain... ... ...) > 0* - здесь преследуется цель выявить форматирующие теги, причем не какие-нибудь, а именно те, которые были переданы в качестве аргументов, и в случае обнаружения - удаляет их (функция Replace с присвоением результатов). Короче говоря, так же поступает и MS Word, когда в выделенном участке текста попадаются и "жирные" участки, и обычные - кликанье по кнопке "B" здесь не сразу приводит к "ожирению"... а только со второго щелчка:).

Четвертое: в связи с тем, что язык HTML является самым свободным языком кодирования (форматирования, если Вам угодно) из всех существующих, никто не застрахован от того, что кто-то в тексте вместо <b> введет с клавиатуры <B>. Ваша программа не заметит этот тег, поскольку рассчитывает увидеть нижний регистр. Не забывайте об этом никогда, покуда работаете над парсингом, особенно если заняты обработкой гипертекстов легкого поведения и всегда страхуйтесь либо UCase, либо LCase. Запомните: UCase и LCase не изменяют передаваемый им текст, если только Вы не присваиваете результат переменной-носителю:

```
a = "Abcd"
a = UCase(a)
```

Как видно из листинга, процедура принимает два аргумента: sBegin (открывающий тег) и sEnd (закрывающий). Они же вскоре и сыграют роль при вычислении длины выделения: LStart "напоминает" программуле, с чего все началось, и длина выделения уже готовенького кусочка будет вычислена так:

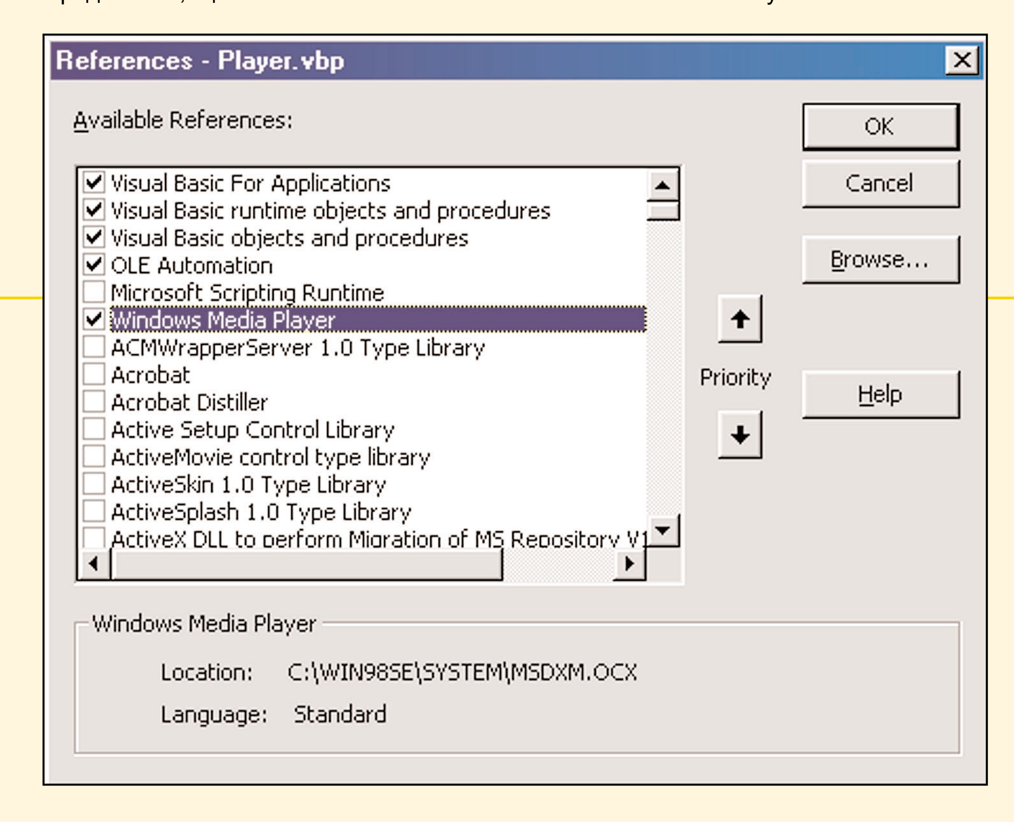
```
LLen + Len(sBegin) + Len(sEnd)
```

для тех, кто на бронепоезде или только-только сел за Бейсик: *[Длина_Первичного_Выделения + Длина_Открывающего_Тега + Длина_Закрывающего_Тега].*

Хоть идея создания текстовго редактора и заманчива, некоторым читателям "Мышления..." все же работа с текстом может покаказаться скучноватой, поэтому иногда мы будет разбавлять "тоску зеленую" интересными "переменками" (периодически, конечно, возвращаясь к MyComPad`у). Тем для рассмотрения - навалом, лишь бы хватало доступной площади на страницах издания.

Сегодня мы создадим собственный MP3-плейер - с плейлистами, паузами, плейями, стопами и регулятором громкости - точь-в-точь как Windows Media Player. При его разработке мы коснемся основных методов работы с элементом управления MediaPlayer, а такие стандартные операции как открытие файла, внесение его полного пути в список (Playlist) были рассмотрены ранее в цикле "Мышление в стиле Visual Basic".

Для начала укажем *Reference* на компонент **MediaPlayer**. Для этого достаточно выбрать из списка Windows Media Player, - основной компонент плейера, а также Microsoft Windows Common Controls - но уже не через диалог References, а в качестве "компонента" (нажмите *Ctrl+T* для вызова списка). Из Common Controls нам нужен только Слайдер (**Slider**, "Ползунок"). те, кто не желает притягивать к проекты лишних ЭУ, может обойтись текстовым полем, в которое будет вводиться числовое значение громкости, либо *HScrollBar`ом*, что более предпочтительно, так как имеет свойства Max, Value и Min, идентичые с свойствами Слайдера. Выглядит, конечно, не так эстетично, зато входит в состав обязательных элементов управления, находящихся в MSVBVM60.DLL. Если Вы решили обойтись подручными средствами, "цеплять" Microsoft Windows Common Controls не нужно.



Для простоты оперирования файлами (в общем-то, их именами) сегодня мы обеспечим их "затаскивание" мышью в список **lstFiles**. Причем процедура Драг-н-Дроппинга будет автоматически определять, тот ли формат файлов был "заброшен"; если да - MP3-файл будет внесен в плей-лист.

На заметку: в рассматриваемом проекте имеют значение не имена файлов из списка, а элементы коллекции, так что вносить в список можно что угодно - вплоть до номеров файлов (например, так: Файл №1, Файл №2, и т.д.).

Для того, чтобы список понял, что в него забросили информацию из файл-менеджера, необходимо его свойство OLEDropMode установить в "1 - Manual". Тогда самое время написать процедуру OLEDragDrop для списка lstFiles (дважды щелкните на списке lstFiles и из списка доступных для этого компонента процедур выберите *lstFiles_OLEDragDrop*):

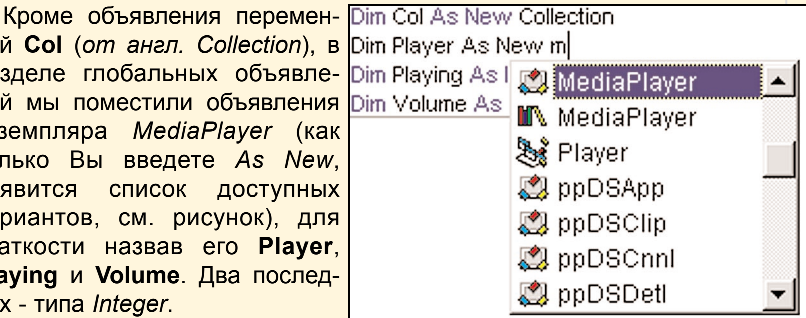
Заполнение именами файлов

```
Private Sub lstFiles_OLEDragDrop(Data As DataObject, Effect As Long, _
        Button As Integer, Shift As Integer, X As Single, Y As Single)
  If Data.Files.Count > 0 Then
    Dim i As Integer
    For i = 1 To Data.Files.Count
      If Right(UCase(Data.Files(i)), 4) = ".MP3" Then
        lstFiles.AddItem Data.Files(i)
        Col.Add Data.Files(i)
      End If
    Next
  End If
End Sub
```

Из рисунка видно, что одновременно с внесением в список строки "Data.Files(i)" коллекция ("Col") поплняется полным путем к файлу, перетаскиваемому из файл-менеджера, причем здесь же проверяется условие: если четыре последних символа в строке, приведенной к верхнему регистру - ".MP3", то файл можно использовать (участок *If...End If*, отмеченный красным). В противном случае он будет игнорироваться. Заметьте: в приведенном листинге отсутствует оператор Else, однако это не значит, что "что-то не сработает" - итератор i (судя по структуре For...Next, отмеченной зеленым цветом).

Итак, имеем коллекцию (или набор - оба понятия имеют равное "хождение"). Чтобы использовать этот набор-коллекцию, нам необходимо ее объявить. Но где?

Давайте разберемся, насколько важна она в рамках проекта (программы), когда должна "рождаться", и когда - "умирать". Поскольку пустая коллекция (т.е. созданная, но еще не содержащая) в принципе много ресурсов не займет, ее создание можно реализовать в разделе глобальных объявлений формы (в данном случае она инициализируется там же, поскольку операторы As New не только создают коллекцию, как, например, *Dim Col As Collection*, но и сообщают Бейсику, что она - ***Новая***, т.е. *попутно и инициализируют*).



Кроме объявления переменной **Col** (*от англ. Collection*), в разделе глобальных объявлений мы поместили объявления экземпляра *MediaPlayer* (как только Вы введете *As New*, появится список доступных вариантов, см. рисунок), для краткости назвав его **Player**, **Playing** и **Volume**. Два последних - типа *Integer*.

```
Dim Col As New Collection
Dim Player As New MediaPlayer.MediaPlayer
Dim Playing As Integer
Dim Volume As Integer
```

Отныне переменная Player - посредник между нашим приложением и библиотекой, к которой Вы еще недавно сделали "референцию". Playing олицетворяет номер трека в коллекции, причем начиная не с нуля, а с единицы. Volume - параметр громкости. Минимальное значение у нас в проекте -1000 (отрицательная тысяча), максимальное - 1000 (положительная).

При нажатии на кнопку Next плейера переменная Playing увеличивается на единицу, при этом MP3Player начинает воспроизведение трека под этим номером.

(Не забывайте: в коллекции индексы не похожи на человеческие, начинаясь с 1, а доступиться к последнему элементу набора можно, в отличие от списков, так: **<name>.Count**.)

В случае, если пользователь нажимает кнопку "Previous" (или как Вы там на ней расписались), Playing уменьшается на единицу, при этом плейер играет меньший номер. Логично?

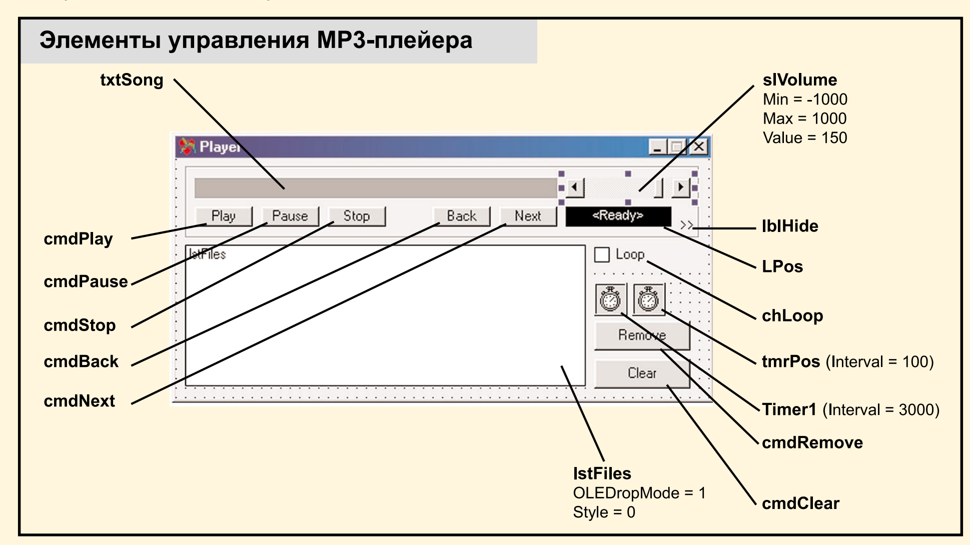
При окончании воспроизведения очередного трека (*Player.PlayState* = *mpStopped*) таймер, установленный на форме с частотой 3000 миллисекунд, опять-таки увеличивает эту "дежурную" переменную на единичку и заставляет Объект Player играть указанный номер в наборе. Однако если трек был последним в списке, то есть Playing - после увеличения его на единицу, - больше количества элементов набора (выделено красным), то сперва проверяется наличие "галочки" у chLoop, и, если она установлена, ход логики перенаправляется к метке M1 (программа "обходит" обозначенный красным участок), а если "галочка" не установлена, таймер отключается, и процедура завершает работу:

Чем занят Таймер?

```
Private Sub Timer1_Timer()
  If Player.PlayState = mpStopped Then
    Playing = Playing + 1
    If Playing > Col.Count Then
          If chLoop.Value = 1 Then
              Playing = 1
              lstFiles.ListIndex = 0
              GoTo M1
          Else
              Timer1.Enabled = False
              Exit Sub
          End If
    End If
M1:
    Player.Open Col.Item(Playing)
    txtSong = Right(Col.Item(Playing), _
      Len(Col.Item(Playing)) - InStrRev(Col.Item(Playing), _
      "\", , vbTextCompare)) ' Здесь можно писать чего угодно
    Me.Caption = "Player - " & txtSong
  End If
End Sub
```

Теперь самое время повносить и расставить элементы управления, при этом, конечно, постараться сделать дизайн удобным и ненагружающим пользователя.

Элементы управления MP3-плейера



При загрузке формы инициализируется громкость Player`a:

```
Volume = Player.Volume
slVolume_Scroll
```

Событие *Scroll* полосы прокрутки достаточно проста в исполнении:

```
Player.Volume = _
   Volume * (1 + (1000 - slVolume.Value) / 100)
```

Как видно, значение (Value) полосы прокрутки, эдакого своеобразного регулятора, вычисляется путем умножения значения переменной Volume на сумму разности максимального значения регулятора и единицы, и все это в конце концов делится на сотню. Сложно? Можно, конечно, и проще: дискретность уровня громкости Player`a и дискретность Слайдера, полосы прокрутки и т.д. отличаются... В принципе, можно "откалибровать" регулятор на глаз - ничего страшного произойти не может, однако лучше сделать как у нас в проекте.

Если Вы уже "прицепили" компонет Windows Media Player к проекту, **Броузер Объектов** покажет Вам все свойства, методы и константы данного компонента. Из всех полезных на данный момент состояний (mpClosed, mpPaused, mpPlaying, mpStopped, mpWaiting) в процедуре для Timer1 нам потребуется только mpClosed, знаменующий конец проигрывания звукового файла. В процедура нажатия кнопки Play рассматривается два случая: проигрывание после полного останова и паузы:

```
Private Sub cmdPlay_Click()
     Playing = 1
     lstFiles.ListIndex = 0
     If Player.PlayState = mpPaused Then
          Player.Play
     Else
          If Col.Count = 0 Then Exit Sub
          Player.Open Col.Item(Playing)
     End If
     Timer1.Enabled = True
txtSong = Right(Col.Item(Playing), _
  Len(Col.Item(Playing)) - InStrRev(Col.Item(Playing), _
  "\", , _
  vbTextCompare))
     Me.Caption = "Player - " & txtSong
     Locking False
End Sub
```

Если есть желание реализовать воспроизведение треков начиная с выделенного в списке и кончая последним, вначале процедуры замените единицу на lstFiles.ListIndex, при этом Playing должен быть равен *lstFiles.ListIndex + 1*. Тогда есть смысл установить единицу для Playing (в списке *lstFiles* это будет *lstFiles.ListIndex = 0*). В таком случае в самое начало процедуры Click кнопки cmdPlay следует внести такой код: *If Col.Count = 0 Then Exit Sub*. Да и в любом случае проверка на наличие элементов в наборе не помешает.

Из приведенного выше кода видно, как вызывается процедура *Locking* с параметрами **False** для ее внутреннего аргумента State. если передать значение True, кнопки очистки списка и удаления текущего элемента станут доступными для "кликанья".

Как же происходит удаление элементов из списка и главное - из коллекции?

В контролах-списках типа *ListBox* и *ComboBox* удаление происходит через вызов метода *RemoveItem*. При этом индексы подразумеваются как Zero based (другими словами, начинающиеся с нуля. Это - стандарт), однако в наборах, как я уже говорил, индексирование начинается с единицы. Оператор, удяляющий элемент из коллекции/набора - *Remove*.

Таким образом, все, что нужно сделать при написании кода для

нажатия на кнопку cmdRemove - это объявить переменную типа *Integer* для итерации посредством структуры *For...Next* - в моем проекте свойство Style списка установлено в **0 - Normal**, однако Вы можете установить его и в **1 - CheckBox** - такой список более "управляем", потому как всегда есть возможность "пробежаться" по нему итератором, например, для проигрывания не всего списка, а только выделенных элементов. За "выделенность" отвечает свойство *lstFiles.Checked* типа *Boolean*, однако не думайте, что если стиль списка - "нормальный", свойство недоступно: разница заключается только в том, что в "нормальном" состоянии список может иметь только один *Checked* элемент - тот, который находтся под фокусом (после щелчка мышью, например), поэтому этот код будет превосходно работать в любом случае - итератор найдет только один выделенный элемент списка.

Для списков со свойством *Style*, установленным в **1**, есть один нюанс: при удалении очередного выделенного элемента *ListCount* изменяется (безусловно), однако об этом ничего не знает переменная итератор ("счетчик действий"), и наверняка попытается добраться до указанного индекса. Естественно, Вы словите ошибку времени выполнения. Чтобы такого не произошло, всегда используют метку перед началом структуры *For...Next* (их, правда, могут разделять различного рода проверки условий и т.д.), таким образом переменная-итератор снова пересчитает количество *ListCount*.

```
Dim i As Integer
Back:
     If lstFiles.SelCount > 0 Then
          For i = 0 To lstFiles.ListCount - 1
               If lstFiles.Selected(i) Then
                    Col.Remove i + 1
                    lstFiles.RemoveItem (i)
                    GoTo Back
               End If
          Next i
     End If
```

Полный "сброс" набора Col также немыслим либо без итерации, либо без *While...Wend* (Вы наверняка помните из предыдущих уроков, в чем принципиальнейшая разница между *For...Next* и этой структурой), потому что наборы по умолчанию не обладают методами типа Clear, а оперируют методами *Remove* (которые применяются только к одному единственному элементу, используя как идентификатор его индекс - либо *Integer*, либо *Long*. В нашем случае - *Integer*): все, что Вам необходимо сделать - это объявить переменную типа Integer, которая пройдет с программой весь цикл осмотра коллекции от первого элемента до последнего (который, кстати, каждый раз будет все меньше и меньше - ровно на единичку). С листбоксом все проще - он имеет специальный метод "очистки" - *Clear*. Поместите этот код в процедуру обработки события *cmdClear_Click*:

```
Dim i As Integer
i = Col.Count
While i > 0
     Col.Remove i
     i = i - 1
Wend
lstFiles.Clear
```

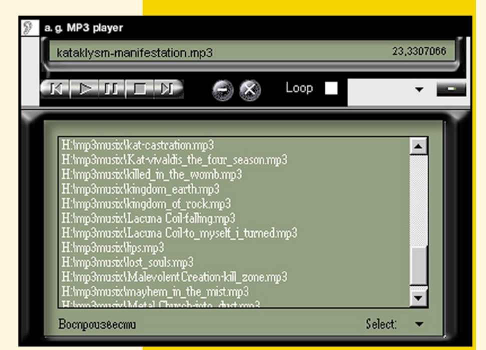
В процедуре нажатия на cmdStop пропишите Player.Stop - стандартную команду Windows media Player`a, вызовите процедуру Locking с положительным аргументом, т.к. свойства Enabled элементов управления cmdRemove и cmdClear теперь должны иметь положительные значения.

Для события "Пауза" главное - разобраться, не поставлен ли уже на паузу плейер - это проверяется через свойство PlayState нашего плейера. Если да - пусть себе мурлычит дальше (Player.Play), а если нет - следует "запаузить" (Player.Pause). А вот здесь необходимо не забыть о таймере: если плейер стоял на паузе, параллельно с запуском воспроизведения нужно и "отпустить" таймер, в противном случае - естественно, наоборот (в обоих случаях используйте его свойство Enabled).

Закрытие главной формы должно сопровождаться удалением ресурсов, занятых уже ненужными процессами. В первую очередь необходимо остановить плейер (даже если он был в состоянии *mpStopped* или *mpPaused* - неважно - ошибки не будет), затем "грохнуть" коллекцию и сам плейер. Player и Col являются Объектами, поэтому применяем операторы Set:

```
Player.Stop
Set Player = Nothing
Set Col = Nothing
```

Вот, собственно, и весь объем работы, связанный с созданием MP3-плейера. Правда ничего сложного?



Вместо эпилога

Есть хорошая идея. Она состоит в том, чтобы проигрыватель при запуске искал MP3-файлы в его же директрии, вносил их в список и, собственно, воспроизводил. Делается это так: в Form_Load создаем локального значения переменную sNextFile типа String и, используя операторы цикла, собираем информацию:

```
sNextFile = Dir(App.Path & "\")
While sNextFile <> ""
     If Right(UCase(sNextFile), 4) = ".MP3" Then
          Col.Add App.Path & "\" & sNextFile
          lstFiles.AddItem sNextFile
     End If
sNextFile = Dir
Wend
If Col.Count > 0 Then
    Playing = 1
    lstFiles.ListIndex = 0
    Call cmdPlay_Click
End If
```

Любители экзотики и просто ПО-эстеты могут скачать мой проект, который выглядит совсем как "железный" аналог, с моей страницы.

Хочу предупредить, что обе программы напропалую используют вызовы API, так что если чувствуете “недопонимание” того, что такое API и основные правила работы с этими интерфейсами, пролистайте старые номера МК (архив “Мышления в стиле...” находится по адресу: http://www.vb.kiev.ua/articles/vbthink/; на этом же сайте находится и сводка API-фукнкций на русском языке), хотя должен заметить, что в работе с вызовами Системных Функций есть только одно основное правило: быть внимательным и не допускать ни орфографических ошибок, ни логических - таких как, например, несоответствие типов передаваемых аргументов - просто следуйте инструкциям, и Ваш Бейсик еще немного поворочается. Ведь, как вы уже, наверное, знаете, при API-сбое “летит” именно IDE Visual Basic. Но мне все же кажется, что Читателю не придет в голову писать вызовы, значения передаваемых аргументов или константы “от фонаря” :-), несмотря на то, что иногда я к этому подтаклкиваю :-).

Чем интересны эти проекты (совмещенные в один, но потенциально автономные), кроме как, собственно, способом выловить “иконки” и появления значка в System Tray`e?

Вот и еще одна переменка в цикле “Мышление в стиле Visual Basic”. Сегодня я порадую Читателя программой извлечения пиктограмм из их носителей (исполняемых файлов и динамических библиотек типа DLL) и реализацию помещения пиктограммы в Системный Лоток (в область индикатора раскладки клавиатуры, часов, индикатора RAS-соединения и т.д.). В принципе, Вам эти приемы в дальнейшем могут понадобиться.

Пара слов о проектах

Во-первых, извлечение пиктограмм из файлов влечет за собой обработку наборов (изображений), поэтому тем, кто только начал интересоваться прикладным программированием на Visual Basic, настоятельно рекомендую прочесть хотя бы ту часть, где рассказывается, как организовать красивые ряды этих пиктограмм - это чистейшей воды логика, никакой математики (разве что сброс переменных в нужных местах). Во-вторых, в проекте также подробно затрагивается “заменитель” CommonDialogs (.ocx) - правда, для экономии места я убрал из него все, что не касается Диалога Открытия Файлов. И еще одна оговорочка: ревнители строгого лексикона а-ля “СуровыйСиПлюсПлюсФорева” и им подобные меня запросто сейчас уличат в своего рода “цинизме” или неграмотности, однако я и сам знаю, кто кому заменитель. Ладно, шутки в сторону - программа использует вызов системной функции GetOpenFileName. Копируйте себе на здоровье, но всегда имейте в виду, что полный листинг объявления этой (и не только) функции ждет Вас на моей страничке.

У проекта извлечения “иконок” есть один маленький недостаток: максимальное количество графических образов, которое может вместиться в графический контейнер, равно его ширине, - и не более того. Конечно, можно сделать окно программы на весь экран, и тогда в PictureBox влезет больше пиктограмм, однако нет гарантий, что в библиотеке Shell32 не появится еще пяток-другой новых. Проблема как раз в том, что объявление функции API, направленной на “примазывание” к картинке полосы прокрутки (ScrollBar) в том виде, в котором его можно считать “работающим кодом”, заняло бы хороших три полосы этого издания. Без моих комментариев - меньше, но не намного...

Посему уловите идею, а в дальнейшем, когда мы коснемся динамических меню или, к примеру, элемент управления ListView, Вы можете воплотить все это другим способом.

Гулять так гулять, подумал я и добавил в этот номер пример “втаскивания” пиктограммы основного окна приложения в Системный Лоток, так что у Вас есть шанс сотворить интересный фокус с Вашей программой (кстати, любой, так как функция может быть использована другими проектами). В чем прелесть этого примера - в том, что “иконка”-то живая! То есть, анимированная. Способ такого “оживления” - весьма примитивный, однако это работает - и все довольны :-).

Типы данных (Пролог)

Чтобы начать написание программы, нам необходимо сперва обеспечить более-менее профессиональный диалог получения имен файлов - не писать же нам его в текстовом поле вручную. Статья начинается с описания проектов. Там же было сказано, что существует некая API-функция, “показывающая” диалоги типа Открыть-Сохранить (функция находится в библиотеке comdlg32). Вот тут и начинается...

Не думалось мне, что придется так - с бухты-барахты рассказывать о **пользовательских типах**... Тема интересна и актуальна, а значит, давайте задумаемся на минуточку, что же это такое.

Бейсик, как и любой другой язык программирования без исключения, различает типы данных, участвующие в процессе выполнения программы. Это не вызывает сомнений. Языки программирования бывают строго типизированными - таких большинство, - и “нестрого”-типизированными. Таковым является Visual Basic (до версии 6.0 включительно): наряду с традиционными типами, - такими как Integer, Boolean, Long, и т.д. - преспокойно существует и тип Variant (См. первые публикации “Мышления в стиле...” или посетите мой сайт). Такой тип может иметь значения всех доступных в VB типов, за редким исключением (в отношении типа Object и других “объектных” применяется несколько другой синтаксис. Читайте позже в разделе об ООП).

В чем “нестроготипизированность” (простите за каламбурчик) Бейсика? А в том, что не объявив явно тип переменной, та, на лету “самообъявляясь” при первом ее вызове или использовании (упоминании, если хотите), - становится “вариантной”. По умолчанию. Хм... Не лишено логики, однако Option Explicit анулирует сию фору - и правильно: Вы просто-напросто можете сделать опечатку в процедуре или функции, модифицирующей эту переменную. Следовательно, “добродушный Бейсик” поймет это по-своему: Вы якобы создали еще одну переменную - теперь в программе есть и strString (которая так и осталась невостребованной всю дорогу, только в конце ее о чем-то “спросили”), и strSting (которая не понятно зачем из шкуры вон лезет, а все напрасно: при раздаче слонов решающее слово за strString или еще какая-нибудь третья - в зависимости от того, сколько ошибок Вы можете сделать в одной программе). Однако вы аправе объявить переменную и так: Dim S. Тогда она действительно станет переменной типа Variant, а Option Explicit к Вам не сможет предъявить претензий :-). Можете поразвлечься: объявите переменную, не указывая ее тип. Затем допишите код вроде этого: MsgBox VarType(S). Вы получите числовые эквиваленты значений. Константы рассмотрены на сайте, и, если все будет окей, возможно, публикации “Мышления...” будут сопровождаться краткими сводками констант, ключевых слов в виде колонки “в тему”. Однако не спешите отчаиваться, если у Вас нет доступа к Сети: объявите все знакомые Вам типы, а затем вызовите окна сообщения для каждой. Учтите, что полученный Вами 0 - тип Variant. Ноль, вероятно, свидетельствует также о том, что это тип по умолчанию.

Ну да ладно, с типами все, вроде, понятно. Это все, что касается “стандартных” типов. Теперь речь пойдет о “нестандартных” -

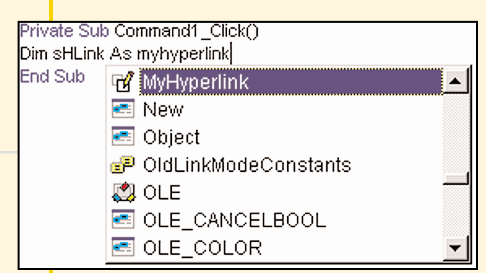
пользовательских. Их иногда называют UDT (User Defiined Types). А раз они носят название пользовательских типов, значит, пользователь может собственноручно “слепить” из того добра, что предлагается комплектом Visual Studio. Заметьте: не только Бейсиковых ресурсов :).

Тогда из чего можно создавать пользовательские типы?

Ответ прост: из всех доступных в данной версии VB типов: String, Integer, Long, Byte, Currency... просто наберите какую-нибудь чепуху в окне редактирования кода в VB IDE, затем добавьте пробел и оператор As - вот из чего могут состоять пользовательские типы. А так как список, который я вас заставил вызвать, динамичен, т.е. при добавлении в проект классов, экземплярчиков Distributed-объектов, и вообще - всего того “объектного”, что уже есть в программе (а это и формы, и элементы управления, и объекты типа Word.Document, и всякая другая дрянь) он пополняется, то... можно сделать вывод, что здесь можно даже заблудиться: когда одна переменная типа UDT состоит из другой UDT, составленной из первой, бpppp... :-)

Теперь серъезнее. В состав идентифицирующих, что ли, компонентов пользовательского типа непременно должны входить другие переменные (как минимум одна) из арсенала, расписанного выше. Например, Ваш тип Hyperlink состоит из Адреса в Интернете, ИмениФайла для сохранения уже локальной закачанной копии файла, ТипПротокола, ИмениПользователя и Пароля (для FTP-серверов это иногда обязательные атрибуты). Перед тем как начать загрузку списка переменных типа Hyperlink Вы обязаны удовлетворить всем условиям для этого типа, иначе ничего у Вас не получится: Бейсик хоть и демократичен, но не настолько... При объявлении типа MyHyperlink необходимо так же явно объявлять и все составляющие его переменные:

```
Public Type MyHyperlink
     Address As String
     FileToSave As String * 255
     Protocol As ProtocolConstants
     UserName As String * 255
     Password As String
End Type
```



После того, как пользовательский тип создан, он становится видимым в Броузере Объектов (Object Browser), а также в контестных списках-подсказках IDE Бейсика. Это значит, чтоВы можете его использовать налево-направо точно так же, как и остальные привычные Вам строчные или целые, логические или денежные (О типах данных читайте на сайте **www.vb.kiev.ua** - там набирает оборотов справочная система по VB 6.0 кроме всего прочего).

Системная функция вывода диалогового окна **GetOpenFileName** использует свой тип - *OPENFILE-NAME*, причем Вы должны создать собственноручно, полагаясь на документацию (См. ранние публикации), который и передается этой функции в качестве аргумента.

```
Private Type OPENFILENAME
     lStructSize As Long
     hwndOwner As Long
     hInstance As Long
     lpstrFilter As String
     lpstrCustomFilter As String
     nMaxCustFilter As Long
     nFilterIndex As Long
     lpstrFile As String
     nMaxFile As Long
     lpstrFileTitle As String
     nMaxFileTitle As Long
     lpstrInitialDir As String
     lpstrTitle As String
     flags As Long
     nFileOffset As Integer
     nFileExtension As Integer
     lpstrDefExt As String
     lCustData As Long
     lpfnHook As Long
     lpTemplateName As String
End Type
```

В отличие от основных синтаксических правил объявлений, здесь Вы не имеете права совмещать строки, например, как при объявлении переменных: Dim strString1 As String, i As Integer и т.д. - все должно идти в столбец и никак иначе. В противном случае вы получите сообщение о синтаксической ошибке внутри блока Type.

А вот и само тело объявления функции:

```
Private Declare Function GetOpenFileName _
     Lib "comdlg32.dll" Alias "GetOpenFileNameA" _
     (pOpenfilename As OPENFILENAME) As Long
```

Прошу заметить, что имя файлика comdlg32.dll можно писать кратко - не указывая ни расширения, ни полного пути - если он - Системный. Говорят, можно не писать и имя... *(Шутка)*

Как говорилось ранее - Вы, конечно, не забыли - при объявлении функций API используется ключевое слово Alias ("псевдоним"). Оно иногда даже спасает при синтаксических накладках, когда Бейсик не позволяет начинать именование функции со знака подчеркивания.

Константы, необходимые для вывода окна диалога (минимальный набор, - все лишнее отсеяно):

```
Public Const OFN_READONLY = &H1
Public Const OFN_HIDEREADONLY = &H4
Public Const OFN_NOVALIDATE = &H100
Public Const OFN_FILEMUSTEXIST = &H1000

Function OpenDialog(Form1 As Form, Filter As _
String, Title As String, InitDir As String) As String
     Dim ofn As OPENFILENAME
     Dim A As Long
     ofn.lStructSize = Len(ofn)
     ofn.hwndOwner = Form1.hWnd
     ofn.hInstance = App.hInstance
 If Right$(Filter, 1) <> "|" Then Filter = Filter + "|"

     For A = 1 To Len(Filter)
       If Mid$(Filter, A, 1) = "|" Then Mid$(Filter, _
       A, 1) = Chr$(0)
     Next

     ofn.lpstrFilter = Filter
     ofn.lpstrFile = Space$(254)
     ofn.nMaxFile = 255
     ofn.lpstrFileTitle = Space$(254)
     ofn.nMaxFileTitle = 255
     ofn.lpstrInitialDir = InitDir
     ofn.lpstrTitle = Title
     ofn.flags = OFN_HIDEREADONLY Or OFN_FILEMUSTEXIST
     A = GetOpenFileName(ofn)

     If (A) Then
          OpenDialog = Trim$(ofn.lpstrFile)
     Else
          OpenDialog = ""
     End If
End Function
```

```
Private Sub cBr_Click()
Dim F As String
F = OpenDialog(Form1, "Executables and DLLs (*.exe, *.dll)|*.exe;*.dll", _
                                                "Choose EXE or DLL", "c:\")
Text1 = Replace(F, Chr(0), "")
If Text1 <> "" Then Command1.Enabled = True
End Sub
```

Теперь собственно об извлечении пиктограмм.

В принципе, Вам потребуется пара строк для достижения цели. В этом преимущество API - за Вас уже многое сделано. Сделано программистами, которые не знают Бейсика, и поэтому все выполнено на Си :-).

В первую очередь давайте разберемся, какие функции выполняют "экстрактирование" иконок. Затем посмотрим, кто же на самом деле их втискивает в ЭУ PictureBox.

Итак, приготовьтесь к шоку:

```
Private Declare Function ExtractIcon Lib _
    "shell32.dll" Alias _
    "ExtractIconA" (ByVal hInst As Long, _
    ByVal lpszExeFileName As String, _
    ByVal nIconIndex As Long) As Long

Private Declare Function DrawIcon Lib "user32" _
    (ByVal hdc As Long, _
    ByVal x As Long, ByVal y As Long, ByVal hIcon _
    As Long) As Long
```

Это и есть те страшные объявления, о которых я говорил...

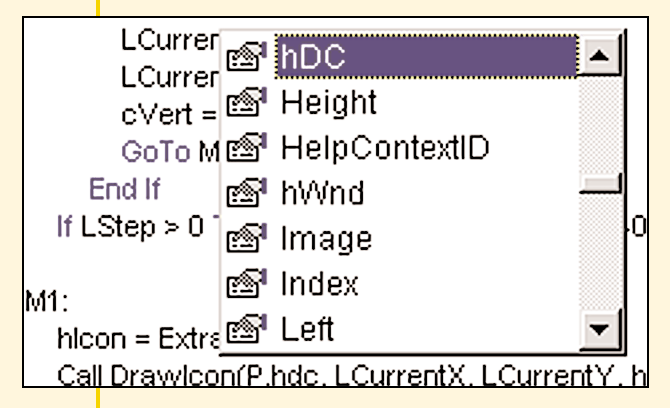
В документации по Win32 API сказано, что функция ExtractIcon извлекает только один значок из файла-контейнера, при этом оперирует переданным ей в качестве аргумента индексом (все значки в таких файлах индексируются при компиляции ресурсов).

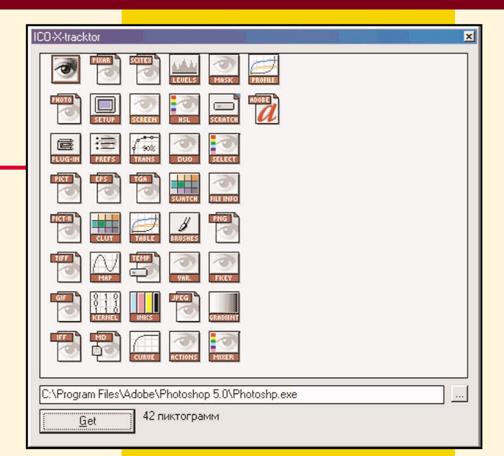
Разберем каждую функцию в отдельности, и начнем с ExtractIcon.

Что принимает эта функция как аргументы?

1. hInst - идентификатор того приложения, которое вызывает функцию. Приведенный мною пример объявляет переменную типа Long, которая так и не изменяет своего значения вплоть до ее использования (т.е. остается равной нулю).

2. lpszExeFileName - переменная типа String, передающая имя файла библиотеки с пиктограммами. Вы наверняка помните публикации о строках в цикле "Мышления"... Нет? Тогда напоминаю: при работе с API и передаче им строк как параметров Вы обязаны убедиться в том, что передаваемая строка оканчивается нулевым символом. Кстати, диалог получения имени файла (OpenDialog), будучи основанным на API-вызове, обеспечивает выполнение этого соглашения должным образом. Однако если пользователь введет путь к файлу в текстовом поле, здесь могут проявятся отличия представления строк VB и Win32. Почему? А потому что вы на самом деле передаете Си-библиотеке не строку, а массив символов (всмотритесь в акт присвоения значения ofn.lpstrFileTitle, например. вы увидите массив из 255 элементов. Не обращайте внимание на разницу в единицу: последний, нулевой символ, также требует места под солнцем :-). А вообще в VB





несколько коряво исполнена идея массивов. На самом деле этих элементов получится не 255, а 256 - не взирая на то,что Вы пишете 255. Об этом - в других статьях. В частности, “Сенсация .NET” бегло пробежалась по недостаткам шестой версии VB), конец которого обозначают нулевым.

3. nIconIndex - переменная типа Long. Используя эту переменную, мы можем узнать сколько там, в библиотеке, пиктограмм, и, узнав, вытащить ту, что нам необходима. Передавая 0, мы “тянем” первую же (как в массиве), а используя отрицательную единицу (-1) - получаем количество.

Что мы получаем от функции ExtractIcon? Посмотрите на конец ее объявления. Видите **As Long**? Это тип получаемого нами результата. Если он равен нулю - в файле для нас ничего нет, в противном случае мы получаем *handle* пиктограммы - своего рода указатель, который мы можем использовать в другой функции - **DrawIcon** для ее отображения.

Разберем по косточкам аргументы этой функции.

hdc - дескриптор контекста устройства (*Device Context Handle*). В примере с отображением вытянутых пиктограмм через **hdc** мы передаем значение свойства **hDC** элемента управления, в котором мы желаем отобразить пиктограмму (Вы можете передать **hDC** формы - тогда пиктограммы будут отображаться непосредственно в окне).

Извлечение всех пиктограмм

```
' L = Количество, cVert = Количество по вертикали,
' LCurrentX = Ряд по горизонтали

Dim L As Long, cVert As Long, LCurrentX As Long, _
LCurrentY As Long, LStep As Long
Dim hIcon As Long, hModule As Long
LCurrentX = 10

L = ExtractIcon(hModule, Text1, -1)   ' здесь мы узнаем количество
If L > 0 Then 'Иконки присутствуют
lblInfo = L & " пиктограмм"
P.Cls
cVert = 0
LCurrentY = 0
  For LStep = 0 To L - 1   ' Цикл от первой иконки до последней
    If cVert >= 8 Then
      LCurrentX = LCurrentX + 40
      LCurrentY = 0
      cVert = 0
      GoTo M1
    End If
  If LStep > 0 Then LCurrentY = LCurrentY + 40

M1:
  hIcon = ExtractIcon(hModule, Text1, LStep)   ' вытаскиваем иконку с индексом LStep
  Call DrawIcon(P.hdc, LCurrentX, LCurrentY, hIcon)   ' и отображаем на картинке P
  cVert = cVert + 1
  Next
Else
  lblInfo = "Пиктограмм нет"
End If
```

Пояснения:

P - элемент управления PictureBox,
Text1 - текстовое поле для ввода имени файла EXE, DLL,
lblInfo - элемент управления Label для отображения текста сообщений

Исходные коды ко многим проектам, рассмотренным в МК, доступны на **www.vb.kiev.ua/code/**

x и **y**, передаваемые функции по ссылке, означают координаты левого верхнего угла пиктограммы в рамках элемента управления, в котором будет происходить обрисовка. Если передать оба значения равными нулю, пиктограмма появится в самом верхнем и самом левом положении. В моем примере используются отступы, равные **10** твипов (я

редко меняю ScaleMode форм и PictureBox`ов) и шагом **40**.

hIcon - дескриптор пиктограммы, которая должна нарисоваться на элементе управления.

Фух... разобрались. Теперь посмотрим, как это выглядит в коде программы.

Наберите этот текст в процедуре нажатия кнопки:

И напоследок пара слов о расположении результатов извлечения динамически. дело в том, что Вы заранее не знаете, сколько потребуется рядов и колонок в массе пиктограмм. Для этого пример использует тот же перебор **For...Next** для увеличения переменной, несущей значение шага, а также переменной, подсчитывающей количество иконок в ряду. Как только лимит для ряда будет достигнут, переменная обнуляется, и в зависимости от этой переменной другие (например, **LCurrentX** и **LCurrentY**) приобретаю новые значения для красивого упорядочения изображений.

Трей



Тема **SysTray** - одна из наиболее популярных. Это и не удивительно: ведь насколько удобно вызывать программы именно оттуда - не из меню кнопки “Пуск”, не из Проводника или командной строки - а из места, которое всегда под рукой, занимает мало места и ... тем не менее, недоступно для большинства программистов на Visual Basic. И вот, несмотря на отличную погоду и летнюю благодать, о которой будем вспоминать осенью и зимой, постараюсь кропотливым тружениками клавиатуры помочь разобраться, что же такое Системный Лоток и как от него добиться хотя бы минимальной функциональности, а именно:

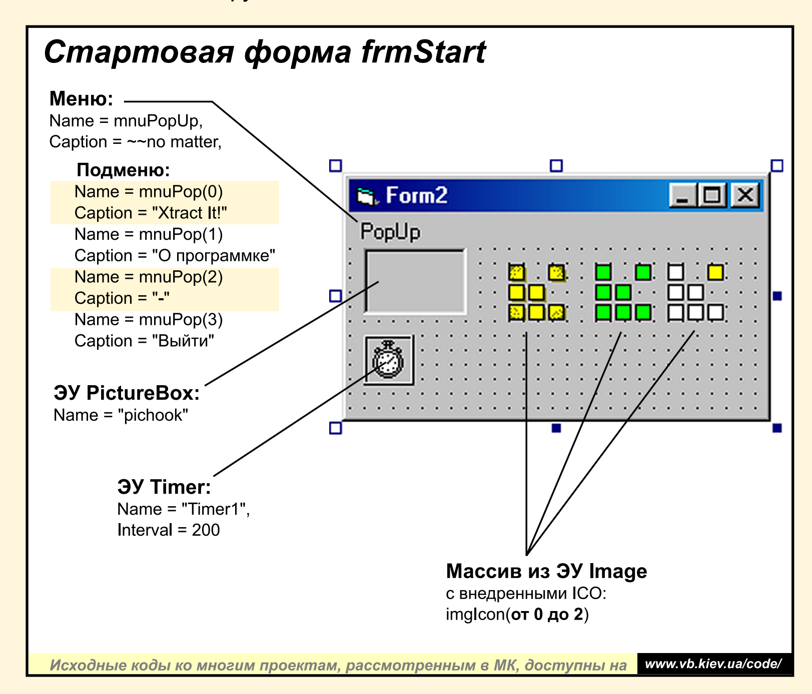
1. Пиктограмма в Трее является анимированным объектом,
2. Из трея можно вызвать окно программы для реализации дальнейших процессов,
3. При наведении курсора на пиктограмму пользователь видит сообщение в виде строки подсказки (ToolTip).

Да, господа, это снова API. Снова дебри констант (которые, к стати, развеиваются, если к этому подойти с “МК” в руках). И, конечно же, немного фантазии.

Начнем-с...

Хочу сразу поставить все точки на i: в этом примере мы используем все тот же ICO-X-traktor ;-), но для того, чтобы все работало на ура, нам необходимо будет, во-первых, добавить еще одну форму, а во-вторых, сделать ее стартовой. Кроме того, набросать на эту новую форму элементов управления (*чем больше тем лучше, причем любых. Ладно. Шутка*). Внизу на картинке показаны компоненты формы frmStart.

Как и положено, начнем с объявления функций, типов и констант.



```
Private Type NOTIFYICONDATA
    cbSize As Long
    hWnd As Long
    uId As Long
    uFlags As Long
    ucallbackMessage As Long
    hIcon As Long
    szTip As String * 64
End Type
Private Const NIM_ADD = &H0
Private Const NIM_MODIFY = &H1
Private Const NIM_DELETE = &H2
Private Const NIF_MESSAGE = &H1
Private Const NIF_ICON = &H2
Private Const NIF_TIP = &H4

Private Const WM_LBUTTONDBLCLK = &H203
Private Const WM_LBUTTONDOWN = &H201
Private Const WM_LBUTTONUP = &H202
Private Const WM_MBUTTONDBLCLK = &H209
Private Const WM_MBUTTONDOWN = &H207
Private Const WM_MBUTTONUP = &H208
Private Const WM_RBUTTONDBLCLK = &H206
Private Const WM_RBUTTONDOWN = &H204
Private Const WM_RBUTTONUP = &H205

Private Declare Function Shell_NotifyIcon _
Lib "shell32" Alias "Shell_NotifyIconA" _
(ByVal dwMessage As Long, pnid As NOTIFYICONDATA) _
As Boolean
Dim TrayI As NOTIFYICONDATA
```

(Вы можете не обьявлять константы для скорости, а просто использовать, например, вместо **NIM_ADD** обычное шестнадцатиричное **&H0**, или **0**)

Надеюсь, все знают, что в модулях (.BAS) переменные локального значения (Private...) не играют сколько-нибудь существенной роли (точнее, могут предназначаться только для нужд этого модуля), поэтому смею предупредить расспросы новичков: раз уж Вы видите тотальную "приватизацию" (звучит! :-)), приведенные выше объявления нужно прописать в разделе General Declarations формы frmStart.

Как Вам всем хорошо известно, ничего просто так не бывает. И пиктограмма не появится в Трее по воле небес. Посему нам нужно занятся процедурой подготовки переменных к работе. Набирая текст объявлений, Вы заметили *UDT NOTIFYICONDATA*. В процедуре **Form_Load** пропишите следующее:

```
    TrayI.cbSize = Len(TrayI)
    TrayI.hWnd = pichook.hWnd
    TrayI.uId = 1&
    TrayI.uFlags = NIF_ICON Or NIF_TIP Or NIF_MESSAGE
    TrayI.ucallbackMessage = WM_LBUTTONDOWN
    TrayI.hIcon = imgIcon(2).Picture
    TrayI.szTip = "Кликни и выползет окно" & Chr$(0)
    Shell_NotifyIcon NIM_ADD, TrayI
    Me.Hide
```

Нужны объяснения? Неужели! :-)

Первое: т.к. UDT - это группа переменных, то необходимо инициализировать ее каждую составляющую. Итак, поскольку многоев

Форточках - массивы, то и многие функции API настолько любознательны, что никак не сработают, не зная всей кухни... что Вы там мне даете? Массив. А какой он длины? Как мне знать, где его окончание? Ага. Вот есть значение.

Ну, не стану тут поясничать - просто намекну, что:

1. “Рубануть” тултипы можно удалив **NIF_TIP** из флагов... Ах да, простите, из **опций**,

2. Все константы, начинающиеся с **WM_**, относят к группе сообщений ОС. Таким образом, покопавшись в winapi.txt, Вы можете соорудить нечто большее, нежели интерактивный интерфейс. (Я как-то совершенно случайно нашел способ избавить окно от надписи (*Caption* = “”), затем, используя API, лишил и кнопок. При этом у меня осталась панель заголовка). Однако не спешите радоваться: обновленный winapi.txt есть у меня на сайте в двух вариантах: от Micosoft (updated) и от Дена Эпплмена (вход с главной страницы - **www.vb.kiev.ua**)... А старый текстовый файл для API Viewer`а полон ошибок и непредвиденностей. Ходят легенды, будто ошибки в API могут вывести из строя монитор, кулер и коврик с мышью...

Вторoе: путем передачи **hWnd** ЭУ PictureBox как параметр **hWnd** для **TrayI**, мы ассоциируем данный контрол с переменной.

Третье: как и любой “*цикл чего-то*” повторяющегося или замкнутого в цепочку, цикл анаонсированной ранее анимации пиктограммы должен с чего-то брать свое начало. Поэтому при загрузке формы зададим начальное изображение: TrayI.hIcon = imgIcon(2).Picture

Четвертое: szTip - не что иное как ТулТип. Если читатель планирует реализовать СисТрей для других целей (например, как пиктограмму быстрого доступа к MyComPad), в строке подсказки можно выводить: количество слов, абзацев, символов в тексте, состояние документа - сохранен/не сохранен, процесс выполнение чего-либо. Например, проверки орфографии. Думаете, нереально? Реально! Стоит только подождать.

Идем дальше: что такое **NIM_ADD**? да ничего особенного... Microsoft ленится объяснять пользователям систему их именования, (а справка по API на MSDN аж кишит подобными вещами). Поэтому приведенный пример именует константу в таком же стиле, дабы попытаться выработать у читателя подобную привычку. Естественно, можно употребить &H0 вместо буквенного удобочитаемого эквивалента - и при этом обойтись без константы:

```
Shell_NotifyIcon &H0, TrayI
```

или даже так:

```
Shell_NotifyIcon 0, TrayI
```

Внимательно читая код, можно увидеть, что три константы, отвечающие за действия над пиктограммой, а именно NIM_ADD, NIM_MODIFY и NIM_DELETE, на самом деле не что иное, как 0,1 и 2. Так, послав сообщение Системе 1, Вы изменяете пиктограмму, а если 2 - удаляете ее из Трея. В процедуре выгрузки формы из памяти используется именно 2 (т.е. &H2), и именно поэтому при останове проекта в IDE VB 6.0 (на стадии разработки) изображение может остаться в Лотке, но при наведении курсора мыши исчезнуть.

Итак, перед тем как написать End в процедуре Form_Unload или Form_QueryUnload (если есть желание переспрашивать пользователя о выходе из программы, в QueryUnload напишите `Cancel = 1`) черканите **Shell_NotifyIcon NIM_DELETE, TrayI**

Как же пиктограмма определяет, какие клики были произведены на картинке? И откуда знает, какие меню отображать?

Заметьте: функция Shell_NotifyIcon пытается нам всучить результат булев... А кому это надо? Ну ладно, если серьезнее, то Вы можете проверить программно, появилась ли иконка. Но учтите: 1. вызов функции в проверке - тоже вызов, так что будьте готовы к ошибке, если напишете дважды; 2. Форточные булевы проделки - это Вам не Басиком баловаться. Другими словами, при возникновении ошибки вы получаете в подарок не True (или False), а 0 ! А правильная проверка выглядит так:

```
If Shell_NotifyIcon(NIM_ADD, _
TrayI) = 0 Then
MsgBox “Cooler Damaged. Repair?”, _
vbYesNo, “Damage notice”
End If
```

Привыкайте к мысли о том, что все в Системе рационально. Кстати, приведенный пример актуален и для простых (простейших) вещей в коде на VB. К примеру, Ваш софт чего-то там долго вычисляет.. Затем необходимо проверить, то ли значение получилось. так вот: сперва присвойте этот результат переменной, а уж затем с нее и спрашивайте. Примерчик в тему:

```
Dim lngFilesCount As Long
lngFilesCount =
GetAllFilesOnDrive(“c:\”)
If GetAllFilesOnDrive(“c:\”) >
1000000 _
   Then MsgBox “Ого, сколько
барахла!”
```

Примерчик ни в дугу, но характеризует специфику ситуации, надеюсь, доходчиво :-)

Ответ прост. Мы используем PictureBox как **ассоциированный** ЭУ. При дабл-клике выполняется одно действие, а при райт-клике - другое. Все, что нам предоставляет PictureBox - все его события, методы и свойства, можно использовать в нужных целях.

Бонус-совет: Если сохранить форму в каталог C:\Program Files\Microsoft Visual Studio\VB98\Template\Forms, - со всеми объявлениями, контролами и кодом - Вы существенно сэкономите время в следующий раз, выбрав шаблон заготовленной формы из списка доступных. Кроме того, сохранив **проект** в подобной директории (\Projects), Вы можете создавать новые проекты на основе практически готовых программ.

Однако в процедуре клика по картинке pichook используются Системные сообщения вместо аргумента Button и только событие **MouseDown**:

```
    Msg = X / Screen.TwipsPerPixelX
    If Msg = WM_LBUTTONDBLCLK Then
        mnuPop_Click 0
     ElseIf Msg = WM_RBUTTONUP Or Msg = WM_LBUTTONUP
Then
        Me.PopupMenu mnuPopUp
    End If
```

В этом случае что левый щелчок, что правый, приведут к появлению меню mnuPopUp, расположенного на стартовой форме. Двойной щелчок выполняет процедуру **mnuPop_Click** с параметром 0, означающим, что “якобы нажато” меню с индексом ноль (не забудьте - это массив). Удалив **Or Msg = WM_LBUTTONUP**, Вы откажетесь от возможности вызывать меню левым одинарным щелчком. Заметьте: mnuPopUp - это меню вернего уровня, содержащее вложенные меню, упакованные в массив (См. схему).

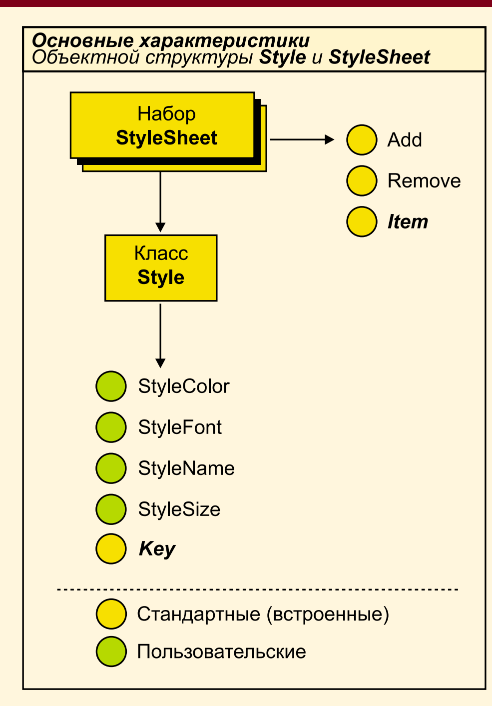
Анимация пиктограммы

Понятное дело, здесь DirectX ни к чему :-) Все “оживление” сведется к поочередному отображению разных картинок. В проекте их три. Вы можете набросать и более - я специально в цикле указал imgIcon.UBound + 1 вместо какого-то определенного числа. Таким образом, вы можете использовать очень много пиктограмм (подойдут только *.ICO), и даже, если больше нечем заняться, сообразить мультипликацию :-)

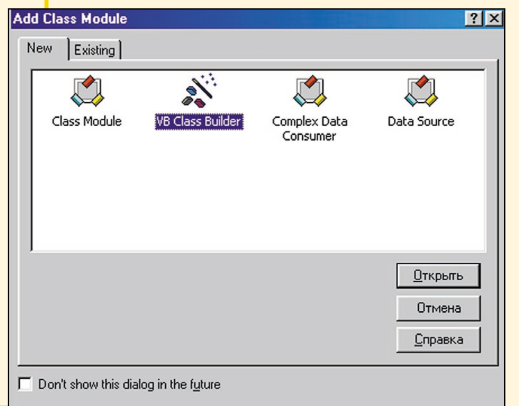
Как это работает? А принцип-то подобен предыдущим! Как говорится, все гениальное - простынь... В нашем же случае используется таймер со скоростью 200 (*См. схему*), изменяющий один из элементов пользовательского типа **NOTIFYICONDATA** (переменная **TrayI**) - правильно: дескриптор изображения, т.е. параметр **hIcon**. Затем посылается сообщение “Изменить_Пиктограмму”... Как только будет достигнуто число, превышающе наибольший индекс в массиве imgIcon, действие переходит на нулевой индекс, то есть первый элемент массива. Для счетчика индексов этих элементов я использую статическую переменную **mPic**. Это значит, что она, являясь даже локальной (чтоб не кушать много ресурсов - таймер все-таки!) сохраняет свое значение до следующего вхождения в процедуру. (Мотайте на ус). И так - каждые 200 миллисекунд.

```
Private Sub Timer1_Timer()
    Static mPic As Integer
    Me.Icon = imgIcon(mPic).Picture
    TrayI.hIcon = imgIcon(mPic).Picture
    mPic = mPic + 1
    If mPic = imgIcon.UBound + 1 Then mPic = 0
    Shell_NotifyIcon NIM_MODIFY, TrayI
End Sub
```

Помнится, начинал я разговор об этой программе с обещанных HTML-возможностей, лже-плагинов и прочей аксессорики... А поскольку большинство задумок так или иначе связаны с ООП (или, по крайней мере, с созданием классов, заданием их свойств и методов - тематики, доселе недостаточно рассмотренной в цикле статей), то решено было отодвинуть рассмотрение их до лучших времен - хотя бы до тех пор, когда читатель станет более-менее свободно себя чувствовать в диалоге с Visual Basic и особенно с Win32 API. Ну вот, шестой круг ада для пользователей Бейсика - API - пройден. Седьмой - ООП - косвенно рассматривался, причем неоднократно, в моих побочных статьях. Например, о VB.NET. Но ведь мы с Вами решили изучить шестую версию Бейсика - самую распространенную, самую, наверное, популярную, исследованную и функционально законченную. Итак, Классы, Объекты... Свойства, методы... Я не стану снова объяснять природу ООП - об этом уже миллион раз сказано в статьях ведущих публицистов, в моих статьях, а также в публикациях моих коллег. Сегодня мы рассмотрим пример использования Class Builder`a - добавки (Add-in) к Visual Basic 6.0, на примере создания класса Style, входящего в набор (Коллекцию) StyleSheet, научимся структурировать объекты, придавать им свойства и назначать методы.

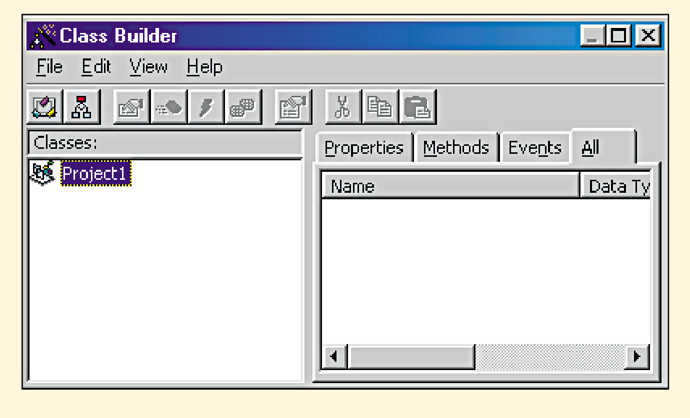


Итак, что такое **Class Builder** и зачем он нужен? Все просто: это добавка к среде разработки, которая упрощает создание Объектов, их групп, включенных в наборы. О своей природе Классы не так сложны, как это может показаться - они имеют свойства, которые определяют их внутренние Property Get, Property Let и Property Set, методы (за это отвечают обычные процедуры и функции), перечислимые типы, пользовательские типы и другие характерные признаки. Естественно, Вас никто не заставляет создавать эти свойства, процедуры, UDT (User Defined Types) и т.д. - всегда следует руководствоваться только нуждами создаваемой Вами программы. Это - искусство, поэтому любые рекомендации здесь сродни советам типа «как написать красивую музыку» или «как научиться рисовать»;-). В наши задачи входит создание Объекта Style, задание ему свойств, присущих элементу таблицы стилей CSS (Cascading Styles Sheet), а также коллекции StyleSheet.

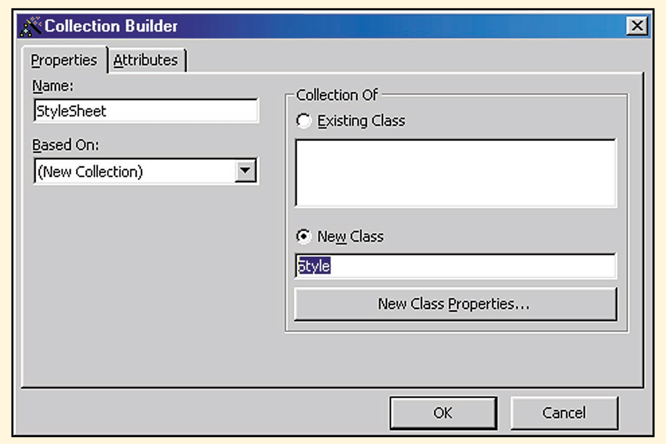


Как и любой VB Add-in, **Class Builder** доступен через меню **Add-Ins>Class Builder Utility**, - если его конфигурация задействует его при запуске IDE. Есть и другой способ: при добавлении модуля Класса в активный проект пользователю показывается характерное диалоговое окно - **Add Class Module**, где Вы вправе выбрать либо шаблон пустого модуля Класса, либо **VB Class Builder** (как показано на рисунке). Либо существующий, сохраненный локально на винчестере. Для этого необходимо кликнуть вкладку Existing. В нашем случае выбираем VB Class Builder.

Утилита в первоначальном состоянии показана на иллюстрации. Как видно, она имеет вкладки Properties, Methods, Events, All - на все случаи жизни. Лично я всегда оставляю All - так легче ориентироваться в структуре Класса.



С помощью кнопок с пиктограммами, изображенными на рисунке, создаем сперва Коллекцию (New Collection) - причем указываем на то, что эта новая коллекция будет состоять из Объектов, созданных параллельно - *не из существующих* - у нас их пока нет (См. рис.).



Там же вводим минимальную информацию о Классе - составляющей Коллекции - имя. Я его назвал Style, а саму Коллекцию - StyleSheet. Новая Коллекция основана на шаблоне обычной Коллекции (Based On: New Collection).

Далее Вашему взгляду предстанет подобная картина: Набор StyleSheet содержит единственный элемент - Style, причем его пиктограмма не является характерной для обычного Класса. Однако разница в очередности создания Классов и их Коллекций, их иерархия, структура зависимостей - это тема отдельной статьи (а ведь об этом я уже писал!), и в данном случае смею заверить: лучшие навыки придут во время практических опытов. Теперь немного по существу: наш главный Объект - Стиль (Style) - «гвоздь» сегодняшней главы - не что иное как способ организовать некое количество субструктур, обладающих многими свойствами. Можно было, конечно, использовать и пользовательские типы, организованные в массив, однако здесь мы имеем маленький нюанс: в дальнейшем читатель сам может дополнить Класс Style дополнительными свойствами, если последует моему примеру, а в случае с UDT это будет практически невозможно, - во всяком случае, очень трудоемко и поэтому нецелесообразно.

Style является составной частью Коллекции **StyleSheet**. Для того, чтобы пополнить список стилей документа, теперь достаточно использовать ключевое слово Add, присущее всем наборам. А так как удобнее всего именовать элементы таких групп по индексам (числам в пределах, ограниченных типом Integer), то следует изменить тип свойства **Key** с *String* на *Integer* (Двойной щелчок на пиктограмме с подписью *Key*, затем выбрать из списка нужный тип). Не знаю, кому удобнее назначать строчные идентификаторы Объектам в Коллекции... Приведенный ниже пример добавления Стиля в набор стилей меня вполне устраивает:




```
StyleSheet.Add StyleSheet.Count + 1, _
    txtName, cboFonts.Text, Val(txtSize), _
    cboColors.ListIndex
```

Схема такова:

```
Добавить_в_коллекцию Новый_элемент, Имя, Шрифт, Кегль, Цвет
```

Как видно из конкретного кода, все параметры берутся из определенных ЭУ, причем некоторые из них - Списки и Комбинированные Списки. Следующий за последним (т.е. Новый) элемент набора определяется прибавлением к последнему единицы. Это естественно: Вы не можете «перепрыгнуть» через один, прибавив двойку, Вы также не можете добавить уже существующий индекс. Для выбора шрифта используется свойство Text Комбинированного Списка cboFonts, заполненного шрифтами динамически (*См. прошлые уроки*). Свойство «Размер» определяется в числовом виде. Тип данных для такого свойства - Integer, поэтому перед употреблением его в коде необходимо преобразование из *String* в числовой эквивалент, используя функцию **Val**. Я бы еще рекомендовал и дополнительную проверку вводимых в txtSize данных: на наличие пробелов, нечисловых символов и т.д. Эта тема рассматривалась ранее.

Для того, чтобы визуальные компоненты преподносили набор как можно более реалистично, отображая уже внесенные стили, их последовательность, вместо текстового поля у меня используется Комбинированный Список - ComboBox. При нажатии кнопки добавить в него добавляется имя стиля:

```
txtName.AddItem StyleSheet.Item(StyleSheet.Count).StyleName
```

Как видно, распознавание Style по индексу - *StyleSheet.Item(<индекс>)* - намного удобнее строчной идентификации. При этом, кстати, возможно и такое обращение к элементу:

```
StyleSheet(<индекс>)
```

stylesheet(
Item(*vntIndexKey As Integer*) As Style

Способов предотвратить внесение в набор стилей с одинаковыми идентификаторами (в нашем случае - именами) существует предостаточно, так что я предоставляю читателю возможность доработать этот вопрос самостоятельно - используя элементарные знания, которые Вы могли приобрести, читая «Мышление», этого нетрудно добиться. Кстати, о поиске в списках: в языке Visual Basic существует настолько удобное словцо InStr... что я подумал: а не сделать ли **InLst**? ... И сделал - MyComPad, который свободен для загрузки с www.vb.kiev.ua, содержит эту функцию. Удобно! J Вы можете использовать ее для поиска элементов в списке стилей.

В коде, приведенном выше, есть маленькая зацепка: элементы добавляются в список, исходя из свойств, описанных в различных элементах управления на форме. Там же, на форме, помещена кнопка «Применить», если пользователь вводит новые значения для выбранного стиля. При нажатии на нее при введенных значениях первого (!) элемента Коллекции происходит генерация ошибки «Несоответствие индекса», другими словами, код события нажатия на кнопку cmdApply

```
StyleSheet.Item(txtName.ListIndex + 1).StyleSize = Val(txtSize.Text)
StyleSheet.Item(txtName.ListIndex + 1).StyleFont = cboFonts.Text
StyleSheet.Item(txtName.ListIndex + 1).StyleName = txtName.Text
StyleSheet.Item(txtName.ListIndex + 1).StyleColor = cboColors.ListIndex,
```

который явно обращается к txtName.ListIndex, не является абсолютно корректным: в списке txtName еще нет элементов, а вводимые в Комбинированный Список данные в него не вносятся. Это следует учитывать. При этом вполне логичным есть решение ограничения доступа к кнопке пользователя в строго определенных случаях. Например, если действие назначения новых параметров стиля применяется к первому элементу набора.

Код для нажатия кнопки cmdAdd:

```
If Trim(txtName) = "" Or _
Trim(txtSize) = "" Or _
IsNumeric(Trim(txtSize)) = False Or _
cboFonts.ListIndex = -1 Then Exit Sub

StyleSheet.Add StyleSheet.Count + 1, txtName, cboFonts.Text, Val(txtSize),
cboColors.ListIndex
txtName.AddItem StyleSheet.Item(StyleSheet.Count).StyleName
cmdApply.Enabled = True
If StyleSheet.Count = 1 Then
     cmdApply.Enabled = False
End If
```

Из кода видно, что сперва необходимо обеспечить выполнение всех требований при внесении стиля в набор: ни одно поле не должно оказаться пустым или некорректно заполненным.

В программе *MyComPad* цветовые заготовки индексируются. Это значит, что свойство Объекта **Style** - **StyleColor** - цифровое выражение (т.е. число), где нулю соответствует черный цвет, единице - белый и т.д. - согласно списку:

0 - Черный
1 - Белый
2 - Серый
3 - Красный
4 - Бордовый
5 - Синий
6 - Зеленый
7 - Желтый
8 - Золотой
9 - Небесный
10 - Водный
11 - Оливка

Можно упростить задачу, упоминая не русские имена цветов, а, например, их «читабельные» HTML-

ярлыки: *Black*, *White*, *Grey*, *Red*, *Maroon* и т.д. Возможно, есть смысл записи их в шестнадцатеричном виде - *#000000*, *#FFFFFF*, *#C0C0C0* - тогда простое использование конфигурационного файла приведет к повышению гибкости программы - Вы дадите возможность пользователю составлять свои списки доступных цветов. В таком случае тип данных свойства **StyleColor** лучше сделать строчным (!) - *String*. Да-да, именно: все равно обозначение цветов будет производиться для гипертекста. Это нетрудно сделать также в коде модуля **Style** и **StyleSheet** - например, в функции Add.

Теперь самое время обеспечить функциональность приведенных фрагментов кода. Создание свойств Объекта заключается во внесении в файл Класса процедур свойств типа *Property Let*, *Property Set*, *Property Get*. Наверное, очевидно, что *Property Get* отвечает за «снятие», считывание информации, а ... *Let* и ... *Set* - за назначение, присвоение с одной лишь разницей: в зависимости от типа данных, хранящихся в этом свойстве, применяется свое ключевое слово. Для всех зарегистрированных в VB типов кроме “объектных” это *Let*. Для встроенных **Collection**, **Object** и т.д. - *Set*. (Таким словом оперируют при присвоении значений и переменным «объектного» типа в любом случае).

При повторном запуске обновленного (после нажатия Update Project - Ctrl+S) утилита Class Builder отобразит структуру проекта - уже с Объектами и Наборами. Добавление свойства Объекту можно свести к щелчку на панели инструментов в этой программе-добавке. Она имеет вид руки с чем-то (наверное, с пачкой свойств J). Впрочем, райт-клик в правой части окна и нажатие соответствующего меню приведет к тому же результату. Разберетесь. То же касается и методов. В нашем случае мы не добавляем собственных методов - они нам ни к чему, а стандартные уже имеются: Коллекции обладают методами Add и Remove - изучайте, их код Вы найдете в модуле Коллекции StyleSheet, когда обновите проект после изменений в окне Class Builder`a.

Для того, чтобы можно было работать с нашими Объектами, мы должны их где-то объявить. Как Вы помните, такого рода Объекты являются весьма важной частью программы (ПО может использовать стили напропалую - лучше использовать единожды созданные стили, чем динамически отнимать у Системы ресурсы), поэтому **Style** и **StyleSheet** должны быть глобально объявленными. Прикол Бейсика в том, что мы можем вообще игнорировать **Style** - достаточно объявления набора **StyleSheet** - добавление может происходить по заданному плану, учитывая свойства Объекта Style как аргументы в процедуре добавления элементов Add.

StyleSheet в стандартном модуле (раздел General) объявляется так:

```
Public StyleSheet As New StyleSheet
```

И все - Коллекция уже создана и готова к использованию.

Теперь, когда мы дали пользователю возможность добавлять стили, редактировать их и удалять (кстати, это действие выполняется аналогично удалению, правда, без указания свойств стиля - с помощью ключевого слова Remove), можно добавлять в документ каскадные таблицы стилей.

Динамики в ряду меню можно достичь, как уже писалось, с помощью создания массива меню, первый элемент которого создается еще во время проектирования. Он может быть либо невидимым, либо иметь совершенно иное предназначение (у меня это меню «<Без стиля>», которое можно использовать для снятия тегов после форматирования строки в редакторе) - как вы еще объясните пользователю пустое меню стиля без надписи? Возможно, есть смысл создавать стиль по умолчанию типа Default или Body с предустановками.

Чтение набора стилей в список происходит в процедуре *GetStyles*:

```
Public Sub getStyles()
If StyleSheet.Count = 0 Then Exit Sub
With frmEditStyle
    Dim i As Integer
    For i = 1 To StyleSheet.Count
        .txtName.AddItem StyleSheet(i).StyleName
    Next
End With
End Sub
```

Обратите внимание на единицу как начало отсчета: элементы наборов нумеруются не от ноля!

Компоновка HTML-кода нас ждет впереди, а в следующем номере мы займемся созданием плагинов для MyComPad`a! Каким образом? Хм... Создадим ActiveX DLL, подгружаемую изнутри редактора, считывающего надпись для меню из DLL и вносимую в массив меню «Plugins»!

Каскадные таблицы стилей в MyComPad. Пояснения

Чтобы сделать возможным более-менее безболезненным создание таблицы стилей в текстовом редакторе MyComPad, нам придется обратиться за помощью к уже известному Вам VB Class Builder`y. Во-первых, создайте набор StyleSheet, причем в диалоговом окне Мастера укажите, что набор этот состоит из новых компонентов - Style. Таким образом Вы активизируете вкладку свойств этого нового Объекта Style, где Вашей задачей станет придание свойств для Style:

```
StyleName
StyleFont
StyleSize
StyleColor
```

В принципе, как Вы этого добъетесь - Ваше личное дело :-)). Важен факт того, что приведенный ниже код будет работать только в том случае, если Ваш Объект **Style** имеет перечисленные свойства.

Добавление нового стиля в StyleSheet, код которого приведен ниже - всего лишь генерированный Мастером фрагмент. Имена переменных соответствуют именованию оных в версии продукта, поставляемого с VB 6.0.

```
Public Function Add(Key As Integer, StyleName _
As String, StyleFont As String, StyleSize As Integer, _
StyleColor As Integer) As Style
    Dim objNewMember As Style
    Set objNewMember = New Style

    objNewMember.Key = Key
    objNewMember.StyleName = StyleName
    objNewMember.StyleFont = StyleFont
    objNewMember.StyleSize = StyleSize
    objNewMember.StyleColor = StyleColor
    mCol.Add objNewMember
    Set Add = objNewMember
    Set objNewMember = Nothing
End Function
```

Вы видите, что процедура добавления нового стиля требует как минимум пяти аргументов, необходимых для описания этого стиля: *Key* - для нумерации/индексирования, *StyleName* - для сохранения свойства Имя, *StyleColor* - для цвета, *StyleFont* - для шрифта (начертания), *StyleSize* - для обозначения размера шрифта. Конечно, это есть наипримитивнейшее описание стиля по системе CSS; более полное описание каскадных стилей Вы всегда найдете в спецификации, а MyComPad, в свою очередь, не претендует на роль профессионального редактора HTML. Хотя, как мне кажется, Вам ничто не мешает задать Объекту Style как стилю текста еще и свойства Bold, Italic, Underlined, StrikeThrough.

Таким образом можно "наклонить" текст с помощью CSS и свойства Font-style:

```
<STYLE>
    MyStyle {font-style:italic}
</STYLE>
```

Жирность определяется уже не Font-style, а Font-weight:

```
<STYLE>
    MyStyle {font-weight:bold}
</STYLE>
```

Возможно, Вы сделаете свою "эдицию" более продвинутой в плане CSS, втиснув еще и BackColor для текста, TetxtAlign (однако видимого эффекта в текстовом поле Вы не добьетесь - только в конечном броузере), возможно, Вас прельстят дополнительные свойства текста, отлично парсируемые абсолютно всеми современными известными броузерами - например, рамки вокруг текста:

```
border, border-bottom, border-bottom-color, border-bottom-style, border-bottom-width, border-color, border-left, border-left-color, border-left-style, border-left-width, border-right, border-right-color, border-right-style, border-right-width, border-style, border-top, border-top-color, border-top-style, border-top-width, border-width.
```

ПРИМЕЧАНИЕ

"Заболдить" текст можно и с помощью ключевого слова **bolder**, и даже указать процентное отношение веса — от 100 до 900, однако здесь Вы, скорее всего, обязательно наткнетесь на проблему "нераспознаваемости" тегов старыми броузерами. Что касается bolder, то здесь старым броузером можно считать и IE 4.

Вы вправе задавать также и поля:

```
margin, margin-bottom, margin-left, margin-right, margin-top.
```

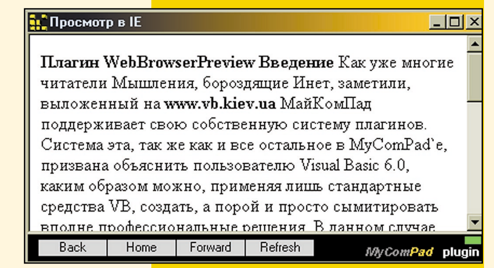
Как эти свойства назовутся в Вашем проекте - дело, опять-таки, только Ваше.

Я же ограничился только шрифтовым описанием стиля, его размером, цветом и именем, предоставив Вам трамплин для изучения CSS.

Еще одно примечание: я частенько переделываю процедуры *Add*, созданные мастером, с одной целью: я всегда знаю, как мне придется обращаться к элементам набора. В данном случае - **StyleSheet**. Я предпочитаю числовые идентификаторы, это прослеживается, наверное, на протяжении всего цикла "Мышления". Поэтому я просто переопределяю тип для переменной *Key* - из *Variant* в *Integer* (в зависимости от ожидаемых объемов коллекции, содержащей этот Объект, я иногда предпочитаю *Long*). Соответственно, все процедуры и функции, добавляющие компонент в набор, также необходимо причесать в области принимаемых аргументов.

Выше был приведен пример уже приготовленной процедуры *Add* из класса **StyleSheet**.

Процедуры добавления стилей, применения изменений в таблице уже введенных, а также нюансы в работе с набором и контексте каскадных стилей были рассмотрены в предущем уроке.



Плагин WebBrowserPreview

Введение

Как уже многие читатели Мышления, бороздящие Инет, заметили, выложенный на **www.vb.kiev.ua** МайКомПад поддерживает свою собственную систему плагинов. Система эта, так же как и все остальное в MyComPad`e, призвана объяснить пользователю Visual Basic 6.0, каким образом можно, применяя лишь стандартные средства VB, создать, а порой и просто сымитировать вполне профессиональные решения. В данном случае мы имеем дело с плагинами, уже зарегистрированными в Системе путем создания инсталятора и, соответственно, инсталляции. Существует великое множество программ-создателей инсталляторов, среди которых я бы мог выделить парочку супернавороченных; некоторые создатели дистрибутивов просто поражают своими "наворотами", другие - попроще, и предлагают лишь джентльменский мимнимум кайфов. Однако сегодня нормой является предоставление пользователю, т.е. автору программы, такой функции как регистрация в Системе разного рода библиотек, ActiveX-экзешников и т.д. Поэтому не будем заострять излишне пристальное внимание тому, КАК зарегистрировать библиотеку, а рассмотрим, как работает самый простой, примитивнейшний механизм плагина.

Первое. Каждый плагин, рассчитанный на какое-либо отдельное приложение, и не являющийся экземпляром плагина утвержденного стандарта, должен как минимум удовлетворять потребностям того приложения, для которого был написан.

Второе. Каждый плагин, даже будучи измененным, модифицированным или сдублированным, должен обеспечивать так называемую "обратную совместимость" - другими словами, если Вы внедряете нововведения в ActiveX DLL с расчетом на такие же новые возможности программы-сервера (ну, это громко сказано, наш МайКомПад не является сервером в прямом понимании этого термина. Тема OLE будет рассмотрена позже), не забудьте, что пользователь может не обновить программу-пользователя плагина. В том случае, если программа не будет готова встретить неожиданности на пути к применению сбойного плагина, результатом ошибки может стать всемирная катастрофа, или, что еще логичнее, использование какого-то Notepad.exe вместо MyComPad`a.

Третье, заключительное. Для стопроцентного обеспечения бессбойной связи MyComPad`a с библиотеками рекомендуется модифицировать лишь внутренние процедуры, собственно обрабатывающие информацию и порождающие какие-либо внутренние процессы. Внешние же лучше оставить неизменными, т.е. "стандартными".

Детали

Начну с того, что все действия и операции внутри программы я привык выносить в автономные процедуры и функции. Зачем? Все просто: во-первых, Ваш код становится намного более читабельным, его проще и приятнее отлаживать в случае чего, и второе - это возможность любого рода автоматизации: позже мы поговорим о написании макросов в МайКомПаде. А сейчас впишите в Form_Load главной формы frmMain следующее:

```
Call GetPlugins
```

Таким образом, мы обеспечили считывание информации о плагинах при запуске приложения. Теперь можем спокойно заняться написанием процедуры считывания. Все что нам нужно - текстовый файл Plugins.prf

в той же директории, что и сам исполняемый файл текстового редактора. К чем у я веду: вместо указания полного пути к файлу установок плагинов мы можем обращаться к нему так:

```
App.Path & "\Plugins.prf"
```

Заметьте: вконце **App.Path** нет слеша, поэтому имя самого файла мы начинаем именно с него.

Исходный код процедуры приведен ниже:

НА ЗАМЕТКУ

В тех случаях, когда файл, открываемый приложением, лежит в одной директории с этим приложением, можно пренебречь **App.Path &** и указывать имя файла без слеша. Однако после **ChDir** (сменить текущую директорию) ситуация меняется не в Вашу пользу.

```
Public Sub GetPlugins()
Dim strLine As String
Dim i As Integer, iCNT As Integer
i = FreeFile
On Error GoTo M2
Open App.Path & "\Plugins.prf" For Input As #i
  While Not EOF(i)
  Line Input #i, strLine
  If Trim(strLine) <> "" And _
        Left(Trim(UCase(strLine)), 9) <> _
        "DISABLED " Then
    On Error GoTo M1
    'Ошибка в файле - такой класс не зарегистрирован
    iCNT = frmMain.mnuPLUG.UBound + 1
    Set PLUGIN(iCNT) = CreateObject(strLine)
    With frmMain
    Load .mnuPLUG(iCNT)
    .mnuPLUG(iCNT).Caption = PLUGIN(iCNT).plgCaption
    If .mnuPLUG.UBound > 0 Then .mnuPLUG(0).Visible = True
    .mnuPLUG(iCNT).Visible = True
    End With
   End If
   iCNT = iCNT + 1
M1: 'Error - нет таких объектов!
   Wend
Close #i
Exit Sub

M2:
Beep
MsgBox Err.Description & vbCrLf & _
     "no Plugins are available", _
     vbOKOnly, "Error"
End Sub
```

Все, что нужно понять: выполняется открытие *App.Path & "\Plugins.prf"* с построчным считыванием до тех пор, пока не наступит конец файла. Этому предшествует обеспечение перехода логики программы на отмеченное лейблом **M2** место в коде, где пользователю сообщается об отсутствии ожидаемого файла, после чего в силу вступает *End Sub* - всякого рода *Exit Sub* здесь просто не нужны.

Далее. Считанная из файла строка проверяется на наличие слова *DISABLED*, которое, кстати, может быть написано в любом виде, т.е. любым регистром (*Вы можете сменить его на любое другое*). Это обеспечивается предварительным переводом всей считываемой строки в верхний регистр с применением *UCase*. Библиотека, успешно зарегистрированная в ОС и обозначенная в файле как *DISABLED*, будет игнорироваться программой *MyComPad*. В текстовом файле могут оказаться пустые строки, неправильная запись имени Объекта, далее - попытка подцепить нечто ужасающее, рушащее Win9x и т.д. - проверка возлагается на Вас, уважаемый пользователь VB 6.0. Мой софт проверяет лишь, не пустая ли строка, а также уносит ноги в том случае, если класс не зарегистрирован. Это *If Trim(strLine) <> ""*... и *On Error GoTo [имя_метки]* перед *Set PLUGIN(iCNT) = CreateObject(strLine)*. Далее

ожидается (в случае успеха) загрузка очередного элемента массива, в роли которого выступает наш "новоподхваченный" плагин. Как видно из требований к плагинам, все плагины для МайКомПада обладают функциями для извлечения из них надписей для меню plgCaption, все они имеют функцию DoAction, причем начиная с MyComPad`a версии 1.02 - и функцию *plgReturned* (плагины для конвертирования из DOS в Win1251 и обратно обновлены с целью снабдить этой недостающей функцией, причем доработан механизм в самом вызывающем их приложении. Все последующие уже учитывают особенности ситуации и снабжены этим свойством). А теперь - по порядку и с подробностями.

plgCaption

Функция, возввращающая строку для надписи в меню приложения-сервера (ну никак не обойтись без термина :-)). Внешняя.

Реализация (проста до безобразия): *CreateObject* создает Объект, зарегистрированный в ОС. После создания такого экземпляра Вы вправе пользоваться его свойствами, методами и и др. Одним из свойств для чтения есть *plgCaption*.

Класс, внутреннее имя которого Plugin.cls, должен иметь такой код:

```
Public Property Get plgCaption() As String
   plgCaption = "Просмотр в IE"
End Property
```

НА ЗАМЕТКУ

Свойства для чтения реализуются путем "комментирования" или просто удаления процедуры *Let* или *Set* для того свойства, которое вы хотите сделать Read Only. Исходный код, доступный на моем сайте (www.vb.kiev.ua) лишь комментирует эту процедуру — для наглядности.

DoAction

Это — стандартизированная внешняя процедура для всех плагинов нашего редактора. Цель — упростить использование библиотеки, так как весьма неудобным было бы решение именовать результирующие вызовы неизвестно как названных функций: выгоднее вызывать все, что нужно библиотеке уже из *DoAction*:

```
Public Function DoAction(strInput As String) As Boolean
   DoAction = WBPrv(strInput)
End Function
```

WBPrv

Внутренняя функция просмотра содержимого текстового поля txtMain в редакторе путем создания файла и загрузки его в окно WebBrowser`a (окно создается в локальном окне МайКомПада — это не инстанс IE! Код плагина имеет в себе референс на MS InternetControls, он же WebBrowser). Тип возвращаемого результата — *Boolean*, — все по той же причине — для отслеживания результатов работы.

```
Public Function WBPrv(strInput As String) As Boolean
     Dim i As Integer
     i = FreeFile
     Open App.Path & "\~wbprv.html" For Output As #i
         Print #i, strInput
     Close #i
     frmWB.Show
     frmWB.WB.Navigate2 App.Path & "\~wbprv.html"
End Function
```

Как видно, полученная строка забрасывается в файл, после чего показывается форма с помещенным на ней WB (т.е. Web Browser), затем, используя метод Navigate2, открываем локальный файл. В

НА ЗАМЕТКУ

DoAction — первая же функция, получающая инструкции типа "что-куда почему и зачем". Возвращаемый тип данных — *Boolean* — помогает отследить результаты выполнения функции, вернее, попытки ее выполнить. К примеру, если вставить обработчик ошибки в эту функцию, то при передаче всякой ерунды можно сделать вид, что информация не изменена; следовательно, переменная *Dirty* в MyComPad`e останется со значением False (если она таковой оставалась до вызова DoAction из плагина). Функция передает полученные данные другой функции — *WBPrv*.

событии *Form_Unload* формы **frmWB** имеется код удаления файла (*Kill* ......).

Это первый плагин, внутренняя функция которого возвращает не видоизмененную строку, а *True* или *False*, что и послужило безусловным поводом переделать событие "клик" меню плагинов — за работу берется *plgReturned*.

plgReturned

Внешняя функция, указывающая на то, вносится ли результат выполнения DoAction в текстовое поле редактора. К такому решению я пришел, когда получил в True в текстовом поле после просмотра в броузере :-)).

```
Public Property Get plgReturned() As Integer
   plgReturned = 0
End Property
```

Поскольку теперь мы имеем дело с избирательным вызовом, код "клика" меню теперь будет таким:

```
Private Sub mnuPLUG_Click(Index As Integer)
Dim Flag As Integer
Flag = PLUGIN(Index).plgReturned

Select Case Flag
Case Is = 1
txtMain.Text = PLUGIN(Index).DoAction(txtMain.Text)
Case Is = 0
Dim vBool As Boolean
vBool = PLUGIN(Index).DoAction(txtMain.Text)
Case Is = 2
' Возможно, мы еще что-нибудь придумаем :-))
End Select
End Sub
```

НА ЗАМЕТКУ

В коде опредения поведения MyComPad`a я определил получаемые значения от 0 до 2:

0 — Нет, плагин не изменяет (не обрабатывает) текст,

1 — Да, текст, измененный плагином, нужно внести в рабочее поле,

2 — Другие случаи, пока мне неизвестные :-)).

В случае с нулем результат уходит "вникуда" — однако Вы можете манипулировать программой в зависимости от его значения.

И последнее:

Формат файла для считывания имен классов приведен ниже:

```
Dos2Win.PLUGIN
Win2Dos.PLUGIN
WBPrv.PLUGIN

([Filename].PLUGIN)
```

ПРИМЕЧАНИЯ

1. Во время разработки небходимо учитывать разницу в путях к исполняемому файлу МайКомПада. В некоторых случаях софтинка не в состоянии увидеть некоторые плагины, цепляет выборочно и т.д. — используйте компилированный экзешник для окончательных тестов.

2. Хорошей идеей считаю создание файлов инициализации для плагинов — когда они станут сложнее и более гибкими, было бы неплохо сохранять их последние пользовательские установки — на кшталт Adobe Photoshop. Эта функциональная добавка никак не повлияет на выполнение плагина вцелом — вперед, дерзайте! :-))



Пролог

Как известно, любая информация ценна, во-первых, своевременностью, во-вторых - достоверностью. К счастью, время, в котором нам довелось жить, располагает средствами для обеспечения оперативного обмена информацией. Однако со вторым аспектом дела обстоят куда сложнее - даже в мир практически тотальной информатизации проблемы авторизации, конфиденциальности и пр. остаются актуальными. Более того, чем глубже общество входит в мир "компьютеризированного образа жизни", тем острее встают вопросы защиты этой самой информации, ее безопасного хранения и столь же безопасного обмена ею.
Как Вы, вероятно, понимаете, компании-разработчики ПО, и в особенности разработчики Операционных Систем, не могли не позаботиться об использовании наиболее удачных, (возможно, надежных), средств защиты в своих ИТ-продуктах. Так, софтверный гигант №1 и многие почтовые клиентские программы ведущих производителей предоставляют более-менее серъезные гарантии авторизации доступа к хранилищам данных, конфиденциальной переписки. При этом используются уже ставшие на "диком Западе" стандартом, например, цифровые подписи - исходя из все более ярких тенденций развития Веб-бизнеса. Впрочем, Украину также ждет неминуемое "одичание" - готовится проект документа о введении цифровых подписей и, конечно же, их всяческое использование в критически важных сферах. Например, в электронной коммерции, которая уже признана существующей в "ридной неньке Украине".
Однако не стоит строить иллюзий насчет надежности алгоритмов шифрования - математически безукоризненные алгоритмы - всего лишь алгоритмы, и, возможно, ко всякому алгоритму можно найти обратный. Вы можете возразить, будто многие из ныне действующих стандартов, основанные на случайных сочетаниях, считаются надежными и их надежность обоснована математическими умами с мировой известностью. Но ведь не следует забывать, что такое "случайность" в компьютерном контексте. Расшифровка на первый взгляд безнадежно "упрятанного", запакованного и скрепленного семью печатями сообщения - всего лишь дело времени. Мы сейчас не говорим о сроках - но факт остается фактом. А если учесть стремительные темпы роста сегодняшних мегагерцевых показателей, можно прийти к выводу, что вчерашние супернавороченные методы шифрования - завтра-послезавтра пара дней работы Вашего домашнего компьютера с программулей, использующей так называемый "метод грубой силы". В чем он заключается? Да в простом переборе вариантов паролей, ключей и т.д. по словарю. И уже не важно, насколько запутанным и математически грамотным является алгоритм - в ходе перебора вариаций на тему правильного пароля этот алгоритм использоваться не будет вообще - нам важен результат. Он или есть, или его, как говорится, еще нет... Опять-таки - другая сторона медали: при нынешних мощностях машин самые передовые методы шифрования гарантируют "Brute-Force-взлом" за время, исчисляемое сотнями лет... Причем увеличение "битности" на порядок удваивает/утраивает время "расшифровки". (Пример для наших маленьких читателей: сколько чисел можно создать из двух знаков? Правильно - 99. А из трех?) Образно говоря, конечно, однако я едва ли преувеличил...
В кругах специалистов, так или иначе связанных с обеспечением компьютерной безопасности, целостности информации или, к

примеру, распределением неких прав доступа к ней, бытует мнение: чем "стандартнее" средство защиты, тем "стандартнее" есть средство взлома. Вы можете, конечно, не согласиться с этой точкой зрения, ссылаясь на бюджет некоторых гигантов программного обеспечения, - следовательно, в уме прикинуть, каковы расходы на разработку подобных стандартов, - и привести многочисленные примеры использования стандартных средств в сфере массовой элетронной коммерции. Однако не так давно в Интернете появились программы Advanced ... Password Recovery Kit, где вместо троеточия можно подставить такие слова как PDF, Zip и др. Автора "пакета восполнения" (русского, работавшего по контракту в Штатах) постигла не самая завидная участь (НТВ поспешило известить) - законодательство США предусматривает более суровое наказание за махинации, т.н. "хакинг" и другие разновидности незаконных деяний в этой области, нежели российское. В Британии, к слову, это дело подвели под статью "террор"... (Кстати, весьма интересно, каковы эти меры пресечения/наказания после заокеанских событий одиннадцатого сентября?)... Все верно: мир еще раз увидел, что взлом алгоритма - дело времени. Возможно, ларчик открывается проще... с другой стороны...

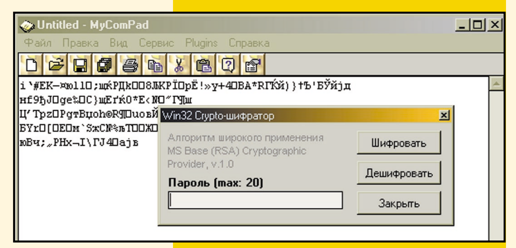
*Однако не будем вдаваться в аллегории или впадать в отчаяние - скорее всего, у Вас не будет столь критических ситуаций, когда цена конфиденциальности будет приближаться к чьим-то сотням бессонных лет (!) или самоотверженному труду коллектива хакеров, наподобие тех, что ломали давеча RC5 :-)). Средствами любого из языков программирования, поддерживающего вызовы Системных Функций (Интерфейс API) для ОС Windows - будь то 95, 98, NT, Me или XP - можно сколотить более-менее криптонадежное приложение в стиле MS Exchange, приложение для защищенного обмена сообщениями на основе системы "Public/Private Keys"... Основа - в API. В данном случае - (в случае с Win 9x + NT) - мы располагаем средством, которым пользуется и Ваш любимый TheBat, и MS Exchange, и Internet Explorer, который "америкосы" юзают в ходе своих онлайн-покупок... И имя ему - **CryptoAPI**.*

Шифрование

Вот и подходит к концу первый период в изучении Visual Basic - Вы, наверное, научились работать с текстом, визуальными компонентами и Объектами в его среде. И сколь нелогичным был путь от самого начала - изучения, что же такое функция и чем она отлична от процедуры или понимания и освоения техники ООП - до создания подобия плагин-системы в VB 6.0, все равно ко мне поступает масса писем, в которых, как обычно, воодушевленный Читатель подкидывает пару-тройку отличных идей. И знаете что самое интересное в этой ситуации? - то, что многие таки повлияли на ход развертывания событий, и вот я в который раз повторяюсь: следующая тема - Интернет-броузер своими руками... Ан-нет - прежде - криптографические приемы «Окон». Того требует наибольшее количество читателей. Однако, прежде чем начать разговор о CryptoAPI, я бы хотел сообщить, что к теме МайКомПада мы еще не раз вернемся (*программка вызвала некоторый интерес не столько у разработчиков, сколько у простых пользователей... Кто б мог подумать???*), - в любом случае многое из того, что будет рассматриваться в дальнейших выпусках, может быть реализовано и в качестве плагина к МКП. Периодически просматривайте сообщения на **www.vb.kiev.ua**, и Вы будете в курсе событий на этом фронте - независимо от того, по какому пути пойдет рассказ о программировании в среде VB 6.0. Кроме того, остались незакрытыми многие темы, как-то: **CSS** (вспоминайте - мы имеем дело с набором *StyleSheet*), экспорт в HTML и др.

Итак, начну вот с чего.

Мне по роду деятельности весьма часто приходится что-либо прятать от глаз людских, что-то, откладывая в долгий ящик, шифровать стандартными и "не очень стандартными" средствами, что-то утаивать от начальства и, кстати, от коллег. Причин много - уверен, Вы тоже не "расшариваете" по сетке свою личную папку с личными текстовками на тему "большой и чистой любви" или с обычными RAS-паролями, базой клиентов своих менеджеров, или, что еще более чудовищно, реальной бухгалтерией :-)) на винчестере с машиной, включенную в общую сеть... Конечно, можно ограничиться шарой лишь одной единственной, публичной папочки, и при этом, как может многим показаться, доступ будет открыт только к ней. Стоп. Не стану долго мучить новичков-пользователей, но это не методы защиты. И даже хранением суперсекретной информации на переносимых драйвах нельзя гарантировать ее сохранность (и даже более того!). Да, существуют программы-затиралки, программы-локеры директорий, дисков и т.д. - однако кто Вам даст голову наотрез, что вся эта система не рухнет вместе с "файлами по работе"? И как Вы представляете отправку по почте защищенного таким образом файлика? :-).



Намного проще зашифровать некоторые файлы с использованием паролей, ключей и пр. - с учетом криптостойкости Оконных API-функций (*читай выше*) Вы можете добиться быстрого (!) доступа к защищенной информации, и притом на 99,9 оставаться уверенным, что, помня пароль, Вы откроете файл завтра.

А почему бы не использовать чужие, готовые разработки из области шифрования целых папок с текстовыми файлами? А потому, что разработав собственный код, можете включить его как плагин к программе, которой пользуетесь ежедневно, и вправе шифровать обрабатываемую информацию не выходя из нее. Удобно!

Что такое CryptoAPI?

Сразу же отмечу - **CryptoAPI** - это типичный для Бейсика "язык" объявлений API, ничуть не проще и не сложнее, нежели функции отправки сообщений окнам или SysTray-иконка... Как и все остальные Си-рожденные Вин-функции, CryptoAPI следует оформлять тем же образом:

```
Public Declare Function WinFunctionName _
     Lib "FileName.ext" Alias "FunctNameA" _
     (FisrstParam As DataType, _
     SecondParam As DataType, _
     ...... _
     ) As DataType
```

Об особенностях типов передаваемых данных я уже как-то говорил. Сейчас я не имею возможности все повторять - См. раздел об API.

Так что же такое **CryptoAPI**? Это набор функций, которыми пользуется некоторое количество программ, входящих в состав ОС Windows 9x плюс еще некоторое неучтенное количество внешних, "левых" типа TheBat (Сорри, конечно...). Со времен Win95 эти функции ОС устанавливались вместе с MS Internet Explorer 3.02 (это одна из причин, по которым в минимальных требованиях того или иного постороннего ПО-продукта прописан IE3.02+). Позднее, с выходом новых "не-серверных" операционок, они лишь совершенствовались, видоизменяясь и видоизменяясь. И вот, видоизмененные, они вошли в состав **ядра** ОС Win 2000. Грубо говоря, серверная часть лагеря Windows должным образом уделяет внимание аспекту CryptoAPI. *Хм... Страшно преположить обратное :-)*.

Что можно сказать в отношении набора этих функций? Хотя бы то, что, предвидя неминуемые изменения в пределах крипт-компонентов, корпорация реализовала сию технологию как гибкую и настраиваемую систему... плагинов... Ну, понятное дело, Вы не найдете в меню кнопки "Пуск" подменю Plugins... (*Неплохая мысль*), однако средствами другой, уже стандартной, функции перечисления этих возможностей вполне реально извлечь данные об ОС и ее криптологическом потенциале. Таким образом, добавление в Систему новых криптослужб не повлияет на выполнение программ, расчитанных на более старые (*или просто другие!*) службы. Новые, как правило, привносятся через Service-Pack`и или с новой версией броузера "ИЕ".

На языке Microsoft каждый из подобных Crypto-компонентов называется **Cryptographic Service Provider (CSP)**. Каждый из "провайдеров" (*вообще-то общепринятым термином является* ***служба****, однако я предпочитаю употреблять первый термин во избежание неминуемых непоняток. Сервис - это один из способов выполнения программы. Крипт-программа также может запущена как сервис, служба на NT-машинке*) предоставляет некие опции - в зависимости от конкретного ал-

горитма. Например, некоторые из алгоритмов принимают (*т.е. функции, отвечающие за данные сервисы и аргументы для них*) нетривиальный набор данных. Или - что более вероятно, - MS пополнила список CSP-провайдеров, причем новые никак не вписываются в класические представления о Crypto-интерфейсе в умах пользователей VB... Вы можете положиться на библиотеку и добросовестность автора и оставить нечто на усмотрение алгоритма, но можете и переопределить дефолт-установки. Читайте ниже. Службы шифрования обладают также и некоторыми свойствами - например, **типом**.

Работа с такими CSP заключается (*в примитиве*) в **открытии контекста CSP**, использовании полученного контекста, и, ясное дело, **закрытии контекста CSP**. Несоблюдение какого-либо из условий этой схемы приведет к ошибке на уровне Системы, так что приготовьтесь к "холодной перезагрузке". Ах, да, - не забудьте сохранять вовремя проекты...

Открытие CSP

Вся канитель с провайдерами шифрования должна начинаться с процедуры открытия контекста выбранного CSP. Это - функция **CryptAcquireContext**.

Вы уже знаете, что в большинстве случаев Система вкладывает в переданные аргументы некоторые результаты. Точно так же работают и CryptoAPI (*в сущности, различать их вовсе и не нужно; природа у них одна*).

Так, **первый** аргумент типа Long будет содержать дескриптор контекста CSP. Этот дескриптор впоследствии и будет использоваться в программном коде как описание того, с чем бы Вы хотели иметь дело (*Description = Описание. Ничего общего с указателями здесь нет*).

Второй аргумент - имя контейнера ключей. Если передана пустая строка (*vbNullString*), "Форточки" будут использовать регистрационное имя пользователя ОС. Настоятельно не рекомендуется использовать такой вариант - мало ли кто/что/зачем изменит имя текущего пользователя сеанса...

Этот аргумент Вам ничего не вернет.

Третьим параметром является имя провайдера CSP. Передавая пустую строку, Вы даете понять Системе, что готовы на ее Default-предустановки. Этот аргумент также не возвращает новых значений. Существуют константы предопределенных имен провайдеров, которые можно передавать в качестве третьего аргумента:

MS_DEF_PROV	Microsoft Base Cryptographic Provider
MS_ENHANCED_PROV	Microsoft Enhanced Cryptographic Provider
MS_DEF_RSA_SIG_PROV	Microsoft RSA Signature Cryptographic Provider
MS_DEF_RSA_CHANNEL_PROV	Microsoft Base RSA Channel Cryptographic Provider
MS_ENHANCED_RSA_CHANNEL_PROV	Microsoft Enhanced RSA Channel Cryptographic Provider
MS_DEF_DSS_PROV	Microsoft Base DSS Cryptographic Provider
MS_DEF_DSS_DH_PROV	Microsoft Base DSS and Diffie-Hellman Cryptographic Provider

Передача в **третий** параметр пустой строки (*Вы уже поняли, что в отношении API это отнюдь не дабл-кавычки!*) означает вариант по умолчанию. В принципе, можно и так - меньше проблем, меньше писанины... А работает не хуже :-)

Четвертым аргументом считают тип провайдера. Возможные варианты приведены в таблице чуть ниже.

Пятым аргументом являются опции открытия контекста. Ниже приведены возможные варианты:

&HF0000000	Без использования личных ключей. Если использован этот флаг, второй параметр *CryptAcquireContext* должен получить vbNullString.
&H8	Создать новый контейнер ключа (значение - во втором параметре!!!, иначе - катастрофа)
&H20	Использовать контейнер в лице имени зарегистрированного пользователя
&H10	Удалить контейнер, указанный во втором параметре функции CryptAcquireContext. Если же во втором параметре пустая строка, будет уничтожен контейнер по умолчанию.
&H40	**Только Win2K (!)** Не использовать UI.

Наконец, сиснтаксис объявления функции:

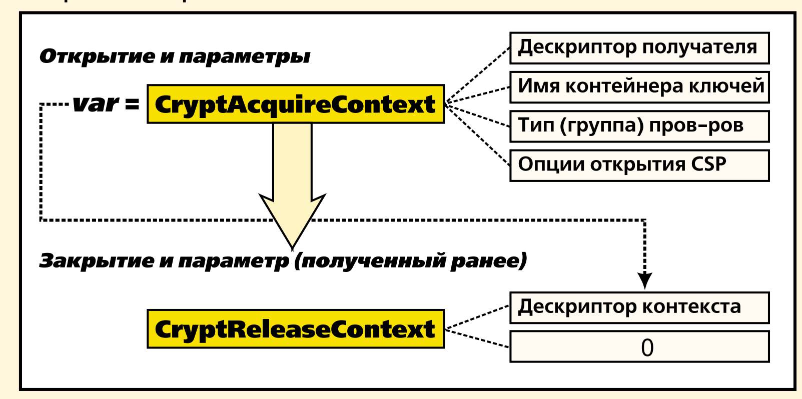
```
Public Declare Function CryptAcquireContext _
   Lib "advapi32.dll" Alias "CryptAcquireContextA"
(phProv As Long, _
   ByVal pszContainer As String, ByVal pszProvider As
String, _
     ByVal dwProvType As Long, ByVal dwFlags As Long)
As Long
```

Закрытие CSP

Закрытие контекста провайдера производится аналогично - просто передайте функции **CryptReleaseContext** значение, полученное от *CryptAcquireContext* (вложенное в первый аргумент-носитель), в качестве второго - **ноль**:

```
Public Declare Function CryptReleaseContext _
     Lib "advapi32.dll" ( _
     ByVal hProv As Long, _
     ByVal dwFlags As Long) As Long
```

Открытие и закрытие контекстов



Перечисление служб (провайдеров)

За перечисление Crypto-служб отвечает **CryptEnumProviders**. Эта функция возвращает данные логического типа (*Boolean* в разрезе VB. Однако при использовании результатов от вызовов API необходимо преобразование типов - используйте *CBool*). Что это значит? Это значит, что, используя указанную функцию и счетчик перечисления, Вы можете убедиться, установлен ли провайдер, соответствующий константе со значением счетчика.

Объявление функции:

```
Public Declare Function CryptEnumProviders _
     Lib "advapi32.dll" _
     Alias "CryptEnumProvidersA" ( _
     ByVal dwIndex As Long, _
     ByVal pdwReserved As Long, _
     ByVal dwFlags As Long, _
     ByVal pdwProvType As Long, _
     ByVal pszProvName As String, _
     ByVal pcbProvName As Long) As Long
```

Передаваемые данные функции должны иметь тип Long, за исключением предпоследнего (**pszProvName**), который, являясь строкой фиксированной длины, заполняется символами с ASCII-кодом 0 - *vbNullString*.

Первым аргументом является **счетчик перечисления**. Как и в массивах в C++, начальное значение - **ноль**.

Второй и **третий** аргументы зарезервированы - передавайте **ноль** и будете спокойны.

Четвертый станет контейнером для переданного **типа провайдера**. Только не думайте, что Вам прямо на блюдечке подадут удобочитаемое описание типа провайдера с автографом CEO MS... Ниже приведена таблица получаемых значений и, соответственно, константы, которым соответствуют эти значения:

PROV_RSA_FULL	1*
PROV_RSA_SIG	2
PROV_DSS	3
PROV_FOTEZZA	4
PROV_MS_EXCHANGE	5
PROV_SSL	6
PROV_RSA_CHANNEL	12
PROV_DSS_DH	13

** **Первый**, в сущности, является наиболее функционально "продвинутым", в отличие от остальных, так или иначе ущемленных в соответствии с запланированными сферами применения: он пригоден и для шифрования, и для формирования цифровой подписи*.

Пятый аргумент будет нести имя провайдера, причем если передана пустая строка (vbNullChar), то шестой передаст длину буфера, т.е. длину той самой фиксированной строки. Если же передана строка известной длины, - Вы обязаны передать в шестом аргументе эту длину, иначе - **ноль**. Тип - **Long**.

Прошлый урок объяснил Вам, что такое *CryptoAPI*, зачем нужна вообще криптография со своими сильными и слабыми сторонами; я попытался рассказать Вам, почему сегодня криптография является оптимальным способом защиты информации - наряду с хард- и софт-изолированием ресурсов. Сегодня мы продолжим знакомиться с API-вызовами функций из разряда криптографических. Понятно, что для работы с данным материалом Вам понадобится, как минимум, набор установленных компонентов. Это либо IE 3.3x, либо Операционная Система семейства MS Windows не ниже Win95 OSR2. Следует учесть также, что не лишним окажется обновление ОС: если у Вас имеется какой-либо MS Win-сервис-пак - самое время найти ему применение. Идеальными условиями является Win98 (лучше - Win2000, WinXP), IE5+.

Итак, теперь мы умеем перечислять провайдеров криптослужб. Однако кроме этого нам может потребоваться знание их типов... Другими словами, требуется перечисление типов провайдеров, доступных (установленных) в Системе. Если перечисление самих крипт-провайдеров производится применением функции CryptEnumProviders, то перечисления их типов можно добиться вызовом очень похожей функции: достаточно в предыдущей функции сменить в вызове несколько слов, как она приобретает иной характер:

```
Public Declare Function CryptEnumProviderTypes _
   Lib "advapi32.dll" Alias "CryptEnumProviderTypesA" ( _
   ByVal dwIndex As Long, _
   ByVal pdwReserved As Long, _
   ByVal dwFlags As Long, _
   pdwProvType As Long, _
   ByVal pszTypeName As String, _
   pcbTypeName As Long) As Long
```

Как видно, функции действительно весьма идентичны.

Обратите внимание на ByVal в списке передаваемых аргументов: pdwProvType и pcbTypeName, единственные параметры, передаваемые по ссылке здесь являются контейнерами для получаемых от функции значений, а переданные по значению аргументы не могут изменены внутри функции (по умолчанию - в случаях без явного указания - VB подразумевает передачу по сылке - ByRef).

Для зарезервированных аргументов по-прежнему передаем нули.

При помощи тех же интерфейсных функций можно узнать, какой из провайдеров назначен для использования в качестве "провайдера по умолчанию". В этом нам поможет **CryptGetDefaultProvider**.

```
Public Declare Function CryptGetDefaultProvider _
Lib "advapi32.dll" Alias "CryptGetDefaultProviderA" (_
ByVal dwProvType As Long, _
ByVal pdwReserved As Long, _
ByVal dwFlags As Long, _
ByVal pszProvName As String, _
pcbProvName As Long) As Long
```

В первом аргументе мы даем наводку CryptoAPI в плане того, среди какого *типа* следует определить Default-провайдера; второй аргумент зарезервирован для будущих поколений Crypto-интерфейсов; третий несет в себе опцию: либо CSP является умолчательным для **текущего пользователя**, либо он является Default **в рамках данной машины** (в первом случае (*юзер*) передаем **&H2**, во втором (*компьютер*) - **&H1**. В документации по Win32API этим значениям назначены константы: CRYPT_MACHINE_DEFAULT = &H1 и CRYPT_USER_DEFAULT = &H2); в четвертом параметре мы получаем <u>**результат**</u> - имя провайдера по умолчанию; в пятом после завершения функции будет храниться **длина** имени провайдера.

Функция возвращает данные логического типа - используйте *CBool* для соответствующего преобразования типов.

Кроме определения провайдера по умолчанию, Вы можете также **назначить** его. Для этого следует передать по значению функции **CryptSetProvider** константы (или их значения) имени провайдера (строкового типа) и его типа (типа Long). В результирующей переменной типа *Boolean* можно отловить результат ее выполнения - опять-таки, через *CBool*.

```
Public Declare Function CryptSetProvider Lib _
    "advapi32.dll" Alias "CryptSetProviderA" (_
    ByVal pszProvName As String, _
    ByVal dwProvType As Long) As Long
```

Кстати, если в качестве первого параметра функции *CryptAcquireContext* (*см. прошлый урок*) передать пустую строку (*запомните - в интерфейсах API используют VB-константу vbNullString!!!*), то будет открыт контекст именно "умолчательного" провайдера.

Сама по себе функция назначения провайдера шифро-сервиса хороша, но не достаточно для "продвинутых" крипто-программ. Здесь у Вас связаны руки по причине отсутствия возможности применения опций. В тех случаях, когда различают Default-провайдеров для текущего пользователя или компьютера (рабочей станции) в целом, а также дополнительные аспекты обращения к этой функции, удобнее использовать CryptSetProviderEx. Эта функция кроме упомянутых аргументов принимает еще два дополнительных: собственно опции (*dwFlags*) и зарезервированный аргумент (*передавайте ноль*).

В качестве опций можно использовать **&H1**, **&H2** или **&H4** - соответствующие им константы в документации озвучены как

```
CRYPT_MACHINE_DEFAULT,
CRYPT_USER_DEFAULT
и CRYPT_DELETE_DEFAULT.
```

Несложно догадаться, что **&H4** (CRYPT_DELETE_DEFAULT) означает удаление (или аннулирование) провайдера по умолчанию.

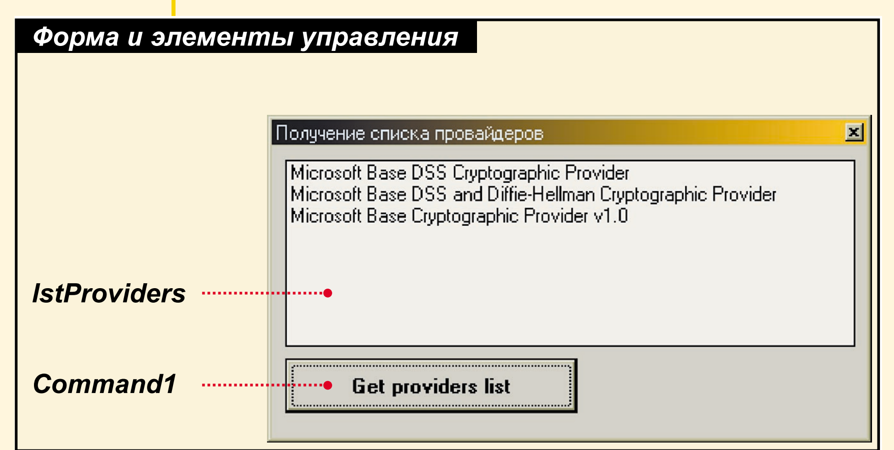
```
Public Declare Function CryptSetProviderEx Lib _
    "advapi32.dll" Alias "CryptSetProviderA" (_
    ByVal pszProvName As String, _
    ByVal dwProvType As Long, _
    ByVal pdwReserved As Long, _
    ByVal dwFlags As Long) As Long
```

Различные типы провайдеров поддерживают различные алгоритмы шифрования (или формирование цифровой подписи)... Очевидный вывод. Естественно, для того, чтобы реализовать в программе тот или иной алгоритм шифрования, необходимо позаботиться о типе провайдера, который может обеспечить программу данным алгоритмом. Ниже приведены типы провайдеров и поддерживаемые ими алгоритмы - в качестве примера взят CryptoAPI в версии, поставляемой с IE5.

Тип	Обмен ключами	Подпись	Шифрование	Хеширование
PROV_RSA_FULL	RSA	RSA	RC2, RC4	MD5, SHA
PROV_RSA_SIG	RSA	MD5	SHA	
PROV_DSS		DSS		MD5
PROV_FORTEZZA	KEA	DSS	Skipjack	SHA
PROV_MS_EXCHANGE	RSA	RSA	CAST	MD5
PROV_SSL	RSA	RSA	Разные	Разные
PROV_RSA_SCHANNEL	RSA	RSA	CYLINK_MEK	MD5, SHA
PROV_DSS_DH	DH	DSS	CYLINK_MEK	MD5, SHA
PROV_DH_SCHANNEL	DH	DSS	RC2, RC4, CYLINK_MEK	MD5, SHA

Вызовы рассмотренных функций

Демонстрацию вызовов упомянутых функций было бы логично начать с перечисления доступных в ОС провайдеров крипто-сервисов. Так и поступим.



Для начала создаем обычный проект VB 6.0 - Standard EXE. Помещаем на главную форму список (ListBox). Имена элементам управления задавайте самостоятельно - на свой вкус, Или же следуйте моему примеру и именуйте компоненты так, чтобы Вы впоследствии были в состоянии что-либо разобрать в коде через неделю-другую. Мой список провайдеров назван lstProviders, имена единственной пары кнопок, нажатие на которые вызывает заполнение списка, выбраны произвольно т.к. не являются принципиальными.

Перечисление провайдеров, как Вы помните, происходит в функции *CryptEnumProviders* - до тех пор, пока она не возвращает логическое *False* (получаемое от преобразования результата через *CBool*), мы получаем информацию о следующем провайдере. Таким образом, введя вызов в цикл, нетрудно добиться формирования целого списка провайдеров:

```
Private Sub Command1_Click()
Dim lResult As Long
Dim lIndex As Long
Dim sNameBuffer As String
Dim lNameLength As Long
Dim lProvType As Long

lIndex = 0
lstProviders.Clear
lResult = CryptEnumProviders(lIndex, 0, 0, lProvType, _
   vbNullString, lNameLength)
sNameBuffer = String(lNameLength, vbNullChar)

While CBool(CryptEnumProviders(lIndex, 0, 0, lProvType, _
                              sNameBuffer, lNameLength))
        lstProviders.AddItem Replace(sNameBuffer, vbCrLf, "")
      lIndex = lIndex + 1
   lResult = CryptEnumProviders(lIndex, 0, 0, lProvType, _
                               vbNullString, lNameLength)
   sNameBuffer = String(lNameLength, vbNullChar)
Wend
End Sub
```

Таким же образом можно построить и код формирования списка **типов** провайдеров - за редким исключением нижеприведенный код повторяет предыдущий:

```
Private Sub Command2_Click()
Dim lResult As Long
Dim lIndex As Long
Dim sNameBuffer As String
Dim lNameLength As Long
Dim lProvType As Long

lIndex = 0
lstProviders.Clear

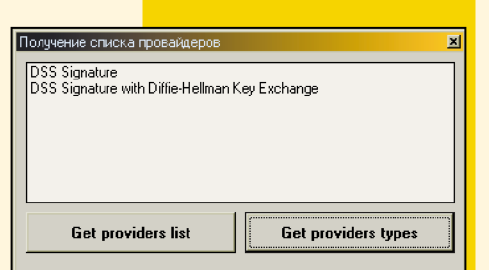
lResult = CryptEnumProviderTypes(lIndex, 0, 0, lProvType, _
    vbNullChar, lNameLength)
lNameLength = lNameLength * 2
sNameBuffer = String(lNameLength, vbNullChar)

While CBool(CryptEnumProviderTypes( _
                lIndex, 0, 0, lProvType, sNameBuffer, _
                lNameLength))
        lstProviders.AddItem sNameBuffer
        lIndex = lIndex + 1
    lResult = CryptEnumProviderTypes( _
    lIndex, 0, 0, lProvType, vbNullChar, lNameLength)
    lNameLength = lNameLength * 2
    sNameBuffer = String(lNameLength, vbNullChar)
Wend
End Sub
```

Хеширование

Еще с незапамятных времен человеку пришла в голову мысль защиты информации, имущества и т. д. при помощи паролей. Возможно, пример окажется не самым удачным, однако ключ и дверной замок все же являются неким подобием системы "пароль-система_доступа". Ну, примеров можно напридумать с дюжину, речь не о способах защиты, а о той концепции "ключ-доступ", которая до боли знакома и встречается нам в быту чуть ли не каждый день.

Как известно, в компьютерных программах довольно часто используются всякого рода защитные "мульки" наподобие серийных номеров программных продуктов, активирующих их ключей и даже пароли доступа к Сети Интернет. Некоторые провайдеры (по крайней мере, киевские) используют собственноручно слепленные программы, генерирующие пароли RAS. Возможно, проще составлять пароли вручную :-). И снова - речь не о способах...



*Полный исходный код простейшей программы, перечисляющей имена провайдеров криптослужб Win32 и их типов, доступен на **www.vb.kiev.ua**.*

Пароль

Что такое пароль - ясно и так. Вы вводите его в нужное место в программе, та, в свою очередь, проверяет его на "правильность", после чего решает, впускать ли Вас в защищенную область. На данном этапе некоторые программисты на Visual Basic допускают весьма грубые оплошности, рассчитывая на незыблемость визуальных компонентов (*все равно - встроенных или внешних*), кода VB и "недалекость" хакеров (*в данном случае я вынужден применить это слово*), которые вдруг загорятся желанием взломать софт. Во-первых, всегда приветствуется сохранение данных в переменных вместо использования ЭУ - видимых или же невидимых во время выполнения. Объясню: поскольку все в Win32 строится на сообщениях, несложно отловить любые свойства любого из компонентов в приложении, равно как и все текущие на

данный момент процессы. Таким образом, у заинтересованного лица на ладони могут оказаться разблокированные "звездочки"-пароли, текст в скрытых полях и т. д. Во-вторых, сама структура защитного блока кода программы должна строиться не на свойствах типа Enabled/Disabled (*вернее, значениях соответствующего свойства - **True/False***), а на функциональных ветвлениях хода выполнения. Например, программу нельзя считать хорошо защищенной только исходя из неактивности кнопки "*Далее*". Например, используя сообщения Windows, можно изменить свойства любого из компонентов программы, в том числе и кнопки "*Далее*" в неграмотно сделанном инсталляторе...

Итак, что такое пароль с точки зрения CryptoAPI?

Да, Вы в чем-то правы: пароль - это Ваш ключ к софтверной парадной двери. Однако он лишь косвенно принимает участие в шифровании/обработке. Как правило, идея пароля состоит в том, чтобы пользователь хранил его в своей памяти, не прибегая к запискам на клочках бумаги, надписям на клавиатуре или сзади монитора, и вводил при необходимости для идентификации. Защита по принципу "да кому все это нужно?!" или "у меня ничего важного нет", конечно, не вызовет особого доверия у тех, кто хотя бы раз оплатил чьи-то прогулки по Интернет :-)... Пароль должен быть "удобозапоминаем" (или, по крайней мере, удобочитаем) и, естественно, уникален. Это главные к нему требования. Таким образом, пароль типа ***Vasya*** никак не подойдет. Однако я знаю множество людей, которые:

- Выбирают в качестве паролей на Yahoo! mail свои "имена наоборот" (*без слов*) :-);

- Используют латинскую раскладку и вводят там свои имена "якобы-по-русски". Неплохо придумано, правда? :-)... Вроде бы, да только существует масса утилит для BruteForce-переборов по словарю, причем многие из них такие штучки уже имеют в виду... Наверное... :-)...

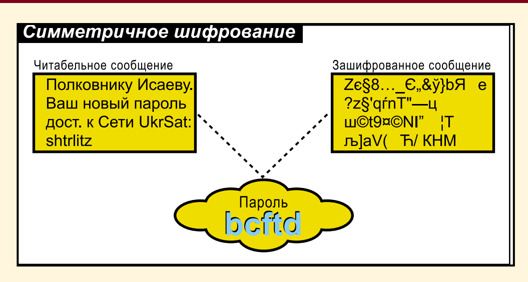
Каков же выход? Как выбирать пароль, чтобы тот оставался читаемым, относительно понятным, и при этом шифрование на его основе давало сколько-нибудь надежный результат? Ведь недостаточно длинный пароль (читай: ключ для алгоритма шифрования) не в состоянии надежно закупорить информацию, а довести его в качестве заполнителя (*например, **vasyavasyavasya***) до необходимой длины мы не можем, так как первый враг шифрования - закономерность... Да и не только поэтому. Все просто, господа: на основе пароля (*можно использовать даже слова из трех букв!*) генерируется - прозрачно для пользователя - хеш, причем в большинстве случаев он (хеш) уже будет состоять из достаточного количества знаков. На основе хеша генерируется пароль. Вы спр**О**сите: ну так что же такое хеш? Хеш - это "временный" (*во многих алгоритмах он действительно, отработав, уничтожается, "забывается", однако нередко хеш является объектом сравнения, цифровыми "отпечатками пальцев" в особо критичных ситуациях. Такой хеш хранится неопределенно долго*) мусор, который генерируется, дабы обеспечить подходящую длину пароля, т. к. в формировании зашифрованного текста будет принимать участие поочередно каждый из элементов обоих массивов символов. Таким образом, длина пароля (хеша) должна хотя бы равняться длине шифруемой информации. Так, по широкоизвестному алгоритму *MD5* слово **Andy** будет иметь такой хеш:

da41bceff97b1cf96078ffb249b3d66e.

Очевидно, что это неподходящий пароль, потому как запомнить его может только гений (правда, не всякий), однако в качестве ключа для шифрования подходит. Не даром хеширование считается основой защиты/проверки/аутентификации во многих системах фильтрации доступа - практически на всех платформах. Таким образом, при вводе пользователем пароля *Andy* в шифрующую функцию поступает *da41bceff97b1cf96078ffb249b3d66e*. Однако на этом положительные стороны хеширования не заканчиваются: генерируя хеш строчной информации, мы можем проверить, достоверна ли она, порой даже не глядя в ее текст. И еще: не обязательно хранить в Системе (на винчестере) "открытый" (*в смысле human-readable*) пароль - достаточно сохранить его хеш, после чего при необходимости сравнивать хеш преподносимой строки с хешем, сохраненным локально. Между прочим, CryptoAPI создает хеш на основании таких параметров, как имя машины, временная метка, отношению веса пыль из-под-кулера к его диаметру, серийный номер ковра для мыши и частоты кадров монитора, деленной на количество непристойностей в Temporary Internet Files (:-)), так что расшифровке хеш не поддается :-) *****.

Один из самых распространенных - из-за скорости выполнения и простоты реализации - методов шифрования является классической парой: "обычный_текст-зашифрованный_текст". Тогда получатель при расшифровке, используя тот же пароль, который был использован для шифрования информации, будет следовать обратной, "зеркальной" схеме: "зашифрованный_текст-обычный_текст". Очевидный факт...

*** Хеш** является результатом необратимого процесса генерирования псевдослучайных (*о компьютерных случайностях я говорил ранее*) чисел, действительно основанный на комбинациях параметров ОС, временной метки и др., отчего считается уникальным производным, и носит название "*односторонний*", т. к. алгоритм его формирования исключает возможность взлома, "расшифровки", - если такой термин здесь вообще уместен.



Такой "стиль" шифрования называют **симметричным**. Это самая простая из всех существующих схем шифрования. Существует еще много интересных концептуальных методов запереть на ключ парадную дверь, - это и шифровальные блокноты, и многочисленные подстановочные таблицы (*классический пример: каждый символ сообщения подлежит подстановке согласно таблицы или списка символов, или согласно арифметическому расчету*), обработка ASCII-массива на n-позиций, элементарный XOR-based алгоритм, шифрование по модулю n и т.д. - однако отнюдь не все дожили до теперешнего времени в виду их несостоятельности в борьбе со взломами. Типичный алгоритм по модулю 26 (*количество латинских кнопочек у Вас на клавиатуре*) мы рассмотрим в ближайшее время как "передышку" между моими CryptoAPI-атаками Читателя :-). Кстати, это и есть основанный на исключении (XOR) алгоритм. А пока попрошу приготовиться к вызову **CryptCreateHash**.

Что, все оказалось настолько просто? - спросит Читатель...

Нет - отвечу я, - все куда проще.. :-)

Итак, приступим к объявлению и вызову функции, формирующей для нас хеш. Она носит имя **CryptCreateHash**.

```
Public Declare Function CryptCreateHash Lib "advapi32.dll" ( _
    ByVal hProv As Long, ByVal Algid As Long, ByVal hKey As Long, _
    ByVal dwFlags As Long, phHash As Long) As Long
```

Первый аргумент, передаваемый этой функции, несет *описатель* (*дескриптор*) контекста CSP, открытый функцией *CryptAcquireContext* (*см. прошлые уроки*); второй представляет собой собственно *алгоритм формирования хеша*. Например, одним из наиболее пропулярных можно считать *MD5*. В таблице ниже приведены доступные (*по состоянию на декабрь 2001 г.*) ID алгоритмов хеширования; третий является дескриптором ключа шифрования (*см. далее*), используемого для работы с алгоритмами *MAC* (*Message Authentication Code*) и *HMAC*; четвертый зарезервирован и ожидает передачу **&H0** (т. е. нуля); **пятый**, **phHash**, станет контейнером для полученного дескриптора хеш-объекта (*см. ниже*). В дальнейшем приложение, оперирующее неким хеш-объектом, будет сталкиваться именно со значением этой переменной.

CALG_MD2	Алгоритм хеширования MD2
CALG_MD4	Алгоритм хеширования MD4
CALG_MD5	Алгоритм хеширования MD5
CALG_SHA	Алгоритм Secure Hash Algorithm
CALG_SHA1	Усовершенствованный алгоритм SHA
CALG_MAC	Алгоритм хеширования Message Authentication Code
CALG_SSL3_SHAMD5	Алгоритм хеширования SSL3 (*Secure Socket Layer 3*), включающий SHA и MD5
CALG_HMAC	Алгоритм хеширования HMAC

Функция **CryptCreateHash** возвращает данные логического типа. Поэтому необходимо использовать функцию *CBool* для преобразования результата в тип *Boolean*.

Собственно искомый результат, - **хеш** - будет храниться в переданной по ссылке переменной **phHash**. Эту переменную и следует использовать в тех функциях, которые оперируют хеш-объектом для шифрования данных.

Что можно сделать с полученным хеш-объектом?

Во-первых, забегая вперед, скажу, что операции с данными на уровне API всегда чреваты передачами их по ссылке в другие функции, так что вполне возможна ситуация, когда Вы можете безвозвратно утерять полученный хеш. Более того, исходя из принципов применения хеш-объектов, логично допустить, что неплохо бы сохранять копию хеша - это зависит от замысла Вашей софтины, а также исходя из некоторых обстоятельств, о которых будет сказано ниже. Итак, копирование хеш-объекта можно произвести путем применения функции **CryptDuplicateHash**. Функция, принимающая четыре параметра-аргумента, два из которых зарезервированы, помещает в один из них новый, сдублированный хеш-объект. Для реализации этой операции требуется как минимум копируемый экземпляр.

Первый аргумент, **hHash**, является тем хеш-объектом, который требуется скопировать. **Четвертый** аргумент, **phHash**, передаваемый по ссылке, будет содержать копию. **Второй** и **третий** лучше не тревожить, - передавайте **нули**.

```
Public Declare Function CryptDuplicateHash Lib "advapi32.dll" ( _
    ByVal hHash As Long, ByVal dwReserved As Long, _
    ByVal dwFlags As Long, phHash As Long) As Long
```

Во-вторых, получение хеш-объекта изначально имеет целью его использование, не так ли? Значит, где-то в двоичных дебрях Advapi32.dll должна жить функция, хеширующая данные на основе этого самого пресловутого хеш-объекта.

Запутались? Объясню: хеш-объект есть описатель, дескриптор некоего объекта, инструкции, если хотите, для хеширующей функции, чтобы та, принимая его в качестве одного из аргументов, имела в виду правила и установки хеширования. Итак, хеширующая функция, собственно выполняющая полезную работу, должна получить дескриптор хеш-объекта, хешируемую информацию (*в формате String*), длину этой информации (*конечно, Long!*), и, если таковые есть, опции. Как Вы уже, вероятно, догадались, опциями здесь называются числовые значения, оформленные как константы типа Long. Еще раз забегу вперед: в данной функции опциям лучше назначить значение **0** (или так: **&H0**).

Вот как все это выглядит в разрезе вызова ее из-под VB 6.0:

```
Public Declare Function CryptHashData Lib "advapi32.dll" ( _
    ByVal hHash As Long, pbData As String, _
    ByVal dwDataLen As Long, ByVal dwFlags As Long) As Long
```

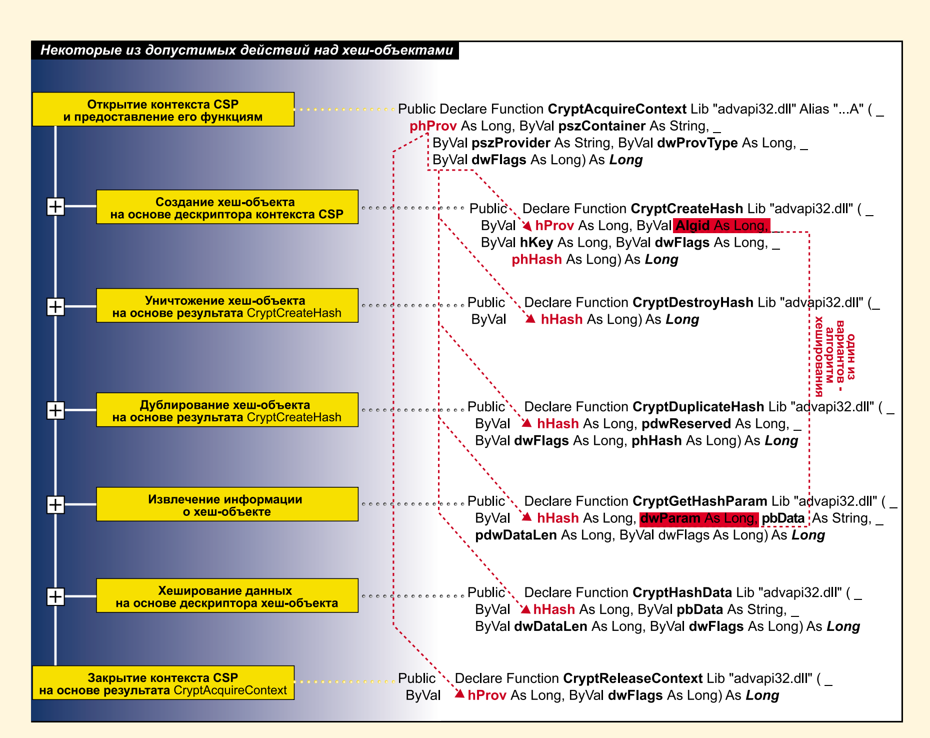
Вот мы и захешировали данные. Да, можно, не прибегая к выявлению текущего значения хеша, передавать куда нужно его дескриптор, однако часто бывает просто необходимо знать его параметры - длину хеш-объекта, текущее значение и алгоритм, которым "замусорили" информацию. Все, чего так хочет функция **CryptGetHashParam**, это переданный по значению дескриптор хеш-объекта (*а по чем еще судить-то?*); параметры, переданные в качестве констант (HP_ALGID = **&H1**, HP_HASHSIZE = **&H4**, HP_HASHVAL = **&H2**) - **1**, **2** или **4**; контейнер для хеша (*передача - по ссылке!, т. е. ByRef, или же просто без явного указания способа передачи*); и **ноль** как последний зарезервированный аргумент. Таким образом, создавая хеш-объект при помощи *CryptCreateHash*, у нас есть возможность узнать, например, каким алгоритмом были захешированы данные, а также текущие данные о самом хеше. Однако здесь есть один нюанс, из-за которого Вам все же, скорее всего, придется сперва дублировать хеш-объект: после извлечения информации хеш-объект "умерщвляется", маркируется как отработанный. Естественно, в целях безопасности. Объявление функции выглядит так:

```
Public Declare Function CryptGetHashParam Lib "advapi32.dll" ( _
    ByVal hHash As Long, ByVal dwParam As Long, _
    ByVal pbData As String, pdwDataLen As Long, _
    ByVal dwFlags As Long) As Long
```

Кроме пречисленных действий над хеш-объектом существует также установка параметров хеш-объекта, которую мы и рассмотрим. На этом тему хешей можно будет считать тщательно рассмотренной.

Ситуация, когда необходимо выполнить установку параметров хеш-объекту, вполне реальна: при попытке хеширования данных алгоритмом с ID *CALG_SSL3_SHAMD5* приходится создавать два дополнительных хеш-объекта, каждый из которых настроен на свой алгоритм, после чего объединенная информация об объектах поступает в функцию. Объявление функции приведено ниже:

```
Public Declare Function CryptSetHashParam Lib "advapi32.dll" ( _
    ByVal hHash As Long, ByVal dwParam As Long, _
    ByVal pbData As String, ByVal dwFlags As Long) As Long
```



Что передается в функцию?

Читатель уже, вероятно, догадался, что по аналогии с другими функциями этого разряда **hHash**, **первый** аргумент представляет собой дескриптор хеш-объекта, который берется не откуда-нибудь, а из переданного по ссылке параметра функции *CryptCreatehash* (*см. иллюстрацию-схему*). **Вторым** параметром является один из вариантов - либо передается значение типа *HMAC_INFO* - для использования алгоритмом *HMAC*, либо устанавливается **значение хеша**. Для первого случая это **&H5**, для второго - **&H2**. **Третий** параметр - это собственно данные, передаваемые для обработки. Если здесь передаются данные об алгоритме кодирования, функцию следует объявлять с использованием псевдонима (*Alias*), причем имя функции звучит как **CryptSetHashDWParam**, в этом случае псевдоним

будет повторять ранее указанное, а третий параметр - **pbData** передается *по ссылке*.

Тип *HMAC_INFO* - тип данных, который Вы обязаны описать в модуле:

```
Type HMAC_INFO
   hashAlgid As Long
   pbInnerString As String
   cbInnerString As Long
   pbOuterString As String
   cbOuterString As Long
End Type
```

Следующий урок *CryptoAPI* будет посвящен **непосредственному использованию хешей** - формированию паролей (или ключей, если выражаться более "технично") для использования в агоритмах симметричного шифрования. Вы узнаете, какие варианты предоставлены программистам при выборе алгоритмов шифровании (провайдер "Microsoft Base CryptoGraphic Provider v. 1.0"), как происходит формирование ключа и какие опции можно выставить, будут рассмотрены вопросы дублирования и уничтожения ключей, а также функции шифрования и расшифровки по Microsoft...



Зайдя на **www.vb.kiev.ua,**
Вы найдете:

- **Книги на тему Visual Basic (5.0, 6.0, .NET),**
- **Исходный код, отсортированный по темам,**
- **Различные библиотеки и другие компоненты многократного использования,**
- **Статьи - полный цикл “Мышления” и другие,**
- **Документацию по Win32 API,**
- **Утилиты и программы.**

Почтовый адрес:

- **Утилиты и программы,**

для отправки проектов для рассмотрения
в статьях, рекомендации и пожелания - **ag@ukr.net**

Последнее обновление: Декабрь 2001